Università
Ca'Foscari
Venezia

Scuola Dottorale di Ateneo
Graduate School

Dottorato di ricerca
in Informatica
Ciclo 29
Anno di discussione 2017

Algorithms for stationary analysis of stochastic Petri nets

SETTORE SCIENTIFICO DISCIPLINARE DI AFFERENZA: INF/01
Tesi di Dottorato di Ivan Stojic, matricola 956114

Coordinatore del Dottorato

Prof. Riccardo Focardi

Supervisori del Dottorando

Prof. Simonetta Balsamo

Prof. Andrea Marin

Università Ca' Foscari Venezia

Dipartimento di Scienze Ambientali, Informatica e Statistica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Algorithms for stationary analysis of stochastic Petri nets

Ivan Stojic

Supervisors

Simonetta Balsamo

Andrea Marin

PhD Coordinator

Riccardo Focardi

September, 2016

Author's web page: http://www.dais.unive.it/~stojic

Author's e-mail: ivan.stojic@unive.it

Author's address:

Dipartimento di Informatica
Università Ca' Foscari di Venezia
Via Torino, 155
30172 Venezia Mestre – Italia
tel. +39 041 2348411
fax. +39 041 2348419
web: http://www.dsi.unive.it

Abstract

Stochastic Petri nets (SPN) are a Markovian formalism for qualitative and quantitative analysis of discrete event dynamic systems. Among other uses, they have been used extensively in performance evaluation of telecommunication systems, computer systems and networks. Analysis of steady-state behaviour of an SPN model usually requires stationary analysis of a continuous-time Markov chain (CTMC) underlying the SPN, whose state space for many practical models is too large to be analysed by direct methods. This serious drawback is shared with many other modeling formalisms and is usually referred to as state space explosion. Usually simulation can be employed to analyse such models. An alternative is to restrict the SPN formalism to product-form SPNs, a class of nets whose unnormalised stationary probability distribution can be obtained in closed form, making stationary analysis much simpler. In this thesis we present algorithms for stationary analysis of SPN models based on efficient encoding of state spaces and transition functions by multi-valued decision diagrams, an efficient data structure. After a short introduction to SPNs and their steady-state analysis, we start with simulation of SPNs and present an algorithm for perfect sampling of SPNs that can be used to directly obtain samples from the stationary distribution. After this, we turn to product-form SPNs and present an algorithm for computation of normalising constant, needed for the normalisation of stationary probabilities in the analysis of product-form models.

Acknowledgements

I would like to thank my supervisors, Simonetta Balsamo and Andrea Marin, for their guidance, advice, support and patience during my PhD at the Ca' Foscari University of Venice; Peter Buchholz and Falko Bause of University of Dortmund for several inspiring discussions; Riccardo Focardi and Nicola Miotello for the impeccable administrative support. Finally, I would like to thank Sanja Singer from the University of Zagreb for the initial impulse to do research.

During the PhD, I have been supported by the Italian Ministry of Education, Universities and Research (MIUR) through the fund *Fondo per il sostegno dei giovani "Programma strategico: ICT e componentistica elettronica"*.

Preface

> By the efforts of other men we are led to contemplate things most lovely that have been unearthed from darkness and brought into light.
>
> — Lucius Annaeus Seneca, *On the Shortness of Life*

This thesis is a result of three years long doctoral studies which were primarily done at the Department of Environmental Sciences, Informatics and Statistics (DAIS) of Ca' Foscari University of Venice, Italy, while a small part of the research was performed during my visit to the Department of Computer Science of University of Dortmund, Germany.

During the studies, I have attempted to improve the state of the art in the analysis algorithms for stochastic Petri nets by considering a class of models with state spaces that are too large to be analysed by standard methods but which can nonetheless be generated and stored in reasonable time and space by using efficient data structures. It seemed to me that for this class of models the full knowledge of structure of the state space and of the underlying stochastic process could be further exploited—despite its prohibitive size—for the stationary analysis of stochastic Petri nets.

Chapter 3, on perfect sampling, is based on three related papers published at the 2015 international conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) [1], the 2016 international conference on Quantitative Evaluation of Systems (QEST) [2] and the 2016 international conference on Performance Evaluation Methodologies and Tools (VALUETOOLS) [3]. Chapter 4, on product-form models, is based on material from a paper [4] submitted in 2016 to the international journal Performance Evaluation that was in the review process at the time of the finalising of this thesis.

Finally, material from two further papers produced during my studies, a paper on modeling and optimisation of an adaptive multiserver system [5], published at the 2014 MASCOTS conference, and a paper on simulation stopping criteria for product-form stochastic Petri nets [6], published at the 2015 EUROSIS European Simulation Multiconference, was not included in the present thesis due to not fitting thematically with the present material.

Contents

1

Introduction

Stochastic Petri nets (SPN) [7, 8, 9] are a widely used Markovian formalism for qualitative and quantitative analysis of discrete event dynamic systems. They have been used for modeling and analysis of various types of systems, such as telecommunications systems, computer systems and networks, manufacturing and biological systems.

In order to perform steady-state performance analysis of an SPN model, an underlying continuous-time Markov chain (CTMC) defined by the SPN model usually needs to be solved (i.e., its stationary probability distribution needs to be computed). Like many other state-based formalisms, in the practical use SPNs suffer from the problem of state space explosion. This problem manifests in an exponential, in model size, growth of the number of states of the underlying CTMC. The state space explosion limits the size of models which can be effectively analysed by standard methods for the analysis of Markov chains (MC); this has resulted in the development of various approximation and solution methods that aim at exploiting the structure of an SPN model or of its underlying CTMC in order to more efficiently analyse the model. In many cases the only feasible approach to the analysis is stochastic simulation, unless the model in question belongs to one of the special classes of models, such as product-form SPNs [10, 11, 12, 13], for which an analytic solution can be efficiently found.

In this thesis we present two algorithms for stationary analysis of SPN models. The algorithms are based on efficient generation and encoding of state spaces and transition functions of SPN models by multi-valued decision diagrams (MDD) [14], a data structure that encodes discrete functions of discrete variables and that has been used successfully to generate very large state spaces of SPN models [15, 16].

The first of the presented algorithms [1] obtains samples directly from the stationary probability distribution of SPN models with finite state spaces, without solving the underlying CTMC; in the literature, this is usually referred to as *perfect simulation* of *perfect sampling*. Like several previous approaches, we base the presented algorithm on a classic perfect sampling algorithm, coupling from the past (CFTP) [17]. The previous approaches [17, 18, 19, 20, 21] implement perfect sampling by restricting the class of models on which they operate and exploit special properties of models in the class to efficiently implement the CFTP algorithm. In contrast, the approach taken in this thesis relies on multi-valued decision diagrams

in order to implement CFTP efficiently for a general class of SPN models, with the only restrictions being that the SPN is required to have finite state space and its transitions are required to have a particular form of marking-dependent firing rates.

The second algorithm [4] presented in this thesis computes a normalising constant for product-from SPNs with finite state spaces. Product-form SPNs are a special class of SPNs for which the stationary probability distribution can be efficiently computed in the form of a product over the components of the SPN model. However, the stationary distribution is obtained in an unnormalised form: the obtained state probabilities do not sum to 1 but to a number $G \in \mathbb{R}_{>0}$, called *normalising constant* (formally, a measure over the state space is obtained). A convolution algorithm [22] for efficient computation of the normalising constant G has been previously developed, but it can only be applied to a special class of SPNs. In contrast, the algorithm proposed in this thesis can compute the normalising constant for a general class of SPNs with finite state spaces, and it can also be used as a basis for algorithms that compute stationary performance measures, again without restrictions on the SPN model other than the finiteness of the state space unlike *mean value analysis* [23], a previous approach that requires the SPN to belong to a special class of models with a specific structure of the state space.

The thesis is structured as follows. First in Chapter 2 we give a short introduction to stochastic Petri nets and define related concepts and properties that are used later in the thesis. We also define multi-valued decision diagrams and introduce some operations on them. Then in Chapter 3 we present the algorithm for perfect sampling in SPNs with finite state spaces and test it on a range of models as well as on a class of SPNs that model fork-join queueing networks (QN), a class of models very useful for the performance evaluation of distributed systems with task concurrency and synchronisation. After this, In Chapter 4 we turn to product-form SPNs and present the algorithm for computation of the normalising constant as well as related algorithms for the computation of performance measures. We test the performance of the proposed algorithm and compare it to the convolution algorithm. Finally, Chapter 5 concludes the thesis.

2

Preliminaries

In this chapter we briefly introduce basic definitions that are used in the later parts of the thesis. First in Section 2.1 stochastic Petri nets are introduced along with some of their qualitative properties. We overview their timed behaviour as well as some of the common stationary performance measures whose computation is one of the main goals of quantitative analysis of SPN models. Then in Section 2.2 we define multi-valued decision diagrams, efficient data structures that we use in the implementation of the algorithms for the stationary analysis of SPNs. We list operations that can be used to create and manipulate these data structures.

2.1 Stochastic Petri nets

Stochastic Petri nets [7, 8, 9] are a formalism used in computer science for modeling and performance evaluation of computer systems, computer networks and telecommunication systems. They have also found use in other fields such as in logistics for modeling of manufacturing systems and in bioinformatics for modeling of biological processes. They are based on untimed Petri nets (PN) [24, 25, 26]—a graphical formalism for the specification and analysis of various types of systems that allows modeling of concurrent, parallel, asynchronous systems—which allows use of the Petri net theory for the qualitative analysis of SPNs. In addition, SPNs are a Markovian formalism—the stochastic process underlying an SPN is a continuous-time Markov chain—which enables use of the rich Markov chain theory for analysis of their timed behaviour. Many extensions of the SPN formalism have been proposed, such as Well Formed stochastic Petri nets [27], Stochastic Activity Networks [28] and Generalized stochastic Petri nets (GSPN) [29, 30]. GSPNs extend SPNs with immediate transitions, achieving the modeling power equivalent to Turing machines [31]. In most cases analysis of a GSPN model involves its reduction to an SPN model, and so with the development of GSPNs the SPN formalism has only gained in importance.

In this section we introduce stochastic Petri nets and briefly overview some of their properties and performance measures that are needed in the later part of the thesis.

2.1.1 Basic definitions

Definition 2.1.1. Stochastic Petri net *is a* 6*-tuple* $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$*, where* $\mathcal{P} = \{P_1, \ldots, P_{N_P}\}$ *is a nonempty set of places,* $\mathcal{T} = \{T_1, \ldots, T_{N_T}\}$ *is a nonempty set of transitions,* $I, O : \mathcal{T} \to \mathbb{N}^{N_P}$ *are transitions' input and output functions,* $W : \mathcal{T} \times \mathbb{N}^{N_P} \to \mathbb{R}_{>0}$ *is a function that defines transition firing rates, and* $\boldsymbol{m}_0 \in \mathbb{N}^{N_P}$ *is an initial marking of the net.*

Fig. 2.1 shows an example SPN with five places depicted as circles and four transitions depicted as bars. Places and transitions are two types of nodes that are connected by arcs in a weighted directed bipartite graph. Functions I and O from the definition of the SPN define the arcs of this graph in the following manner:

- $I_j(T_i) > 0$ if and only if there is an arc from place P_j to transition T_i with the weight $I_j(T_i)$. We say that P_j is an *input place* of transition T_i;
- analogously, $O_j(T_i) > 0$ if and only if there is an arc from transition T_i to place P_j with the weight $O_j(T_i)$. P_j is called an *output place* of transition T_i.

In the example SPN, arcs incident with transition T_1 are defined by $I(T_1) = [1, 0, 0, 0, 0]^\intercal$ and $O(T_1) = [0, 1, 2, 0, 0]^\intercal$.

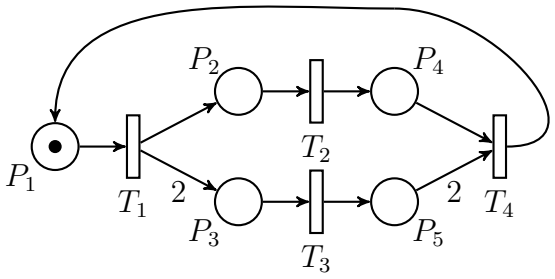


Figure 2.1: An SPN with five places, four transitions and initial marking $\boldsymbol{m}_0 = [1, 0, 0, 0, 0]^\intercal$.

Definition 2.1.2. Marking *of an SPN is a vector* $\boldsymbol{m} = (m_1, \ldots, m_{N_P}) \in \mathbb{N}^{N_P}$ *that associates numbers* $m_1, \ldots, m_{N_P}$ *of* tokens *with places* $P_1, \ldots, P_{N_P}$*, respectively.*

In the graphical representation of SPNs, marking is depicted with black circles inside the corresponding places; alternatively it can be depicted with the number of tokens inscribed inside the places.

Definition 2.1.3. *A transition* $T_i \in \mathcal{T}$ *of an SPN is* enabled *to fire in a marking* $\boldsymbol{m} \in \mathbb{N}^{N_P}$ *if* $\boldsymbol{m} - I(T_i) \geq \boldsymbol{0}$*, i.e., all components of the vector on the left-hand side are nonnegative. We denote the fact that* T_i *is enabled in marking* $\boldsymbol{m}$ *with* $\boldsymbol{m} \xrightarrow{T_i}$*.* Enabling degree $e_i(\boldsymbol{m})$ *of transition* T_i *in marking* $\boldsymbol{m}$ *is defined as* $e_i(\boldsymbol{m}) := \max\{k \in \mathbb{N} : \boldsymbol{m} - kI(T_i) \geq \boldsymbol{0}\}$*. It will become apparent from the discussion that follows that* $e_i(\boldsymbol{m})$ *is equal to the number of times that the transition* T_i *can fire in a row from marking* $\boldsymbol{m}$ *without firing any other transitions in the mean time. A transition* T_i *that is enabled in a marking* $\boldsymbol{m}$ *is* singly enabled *if* $e_i(\boldsymbol{m}) = 1$*, and* multiply enabled *if* $e_i(\boldsymbol{m}) > 1$*.*

In an untimed Petri net, marking process evolves by choosing in a non-deterministic manner one of the enabled transitions and *firing* it. When an enabled transition T_i fires, marking $\boldsymbol{m}$ is transformed to marking $\boldsymbol{m} - I(T_i) + O(T_i)$. We denote the firing of transition T_i with $\boldsymbol{m} \xrightarrow{T_i} \boldsymbol{m} - I(T_i) + O(T_i)$. Similarly, we denote firing of a sequence $\sigma = T_{i_1}, \ldots, T_{i_k}$ of transitions that transforms marking $\boldsymbol{m}$ to a marking $\boldsymbol{m}'$ with $\boldsymbol{m} \xrightarrow{\sigma} \boldsymbol{m}'$. In the example SPN, transition T_1 is the only enabled transition and its firing is an atomic operation that consumes one token from place P_1 and deposits one token in place P_2 and two tokens in place P_3, resulting in the marking $[0, 1, 2, 0, 0]^\intercal$. The set of all markings that are reachable by firing some sequence of (enabled) transitions from the initial marking $\boldsymbol{m}_0$ is called *reachability set* of the SPN and denoted with $\mathcal{RS}$. By also considering the transitions that transform a marking from the reachability set into another marking, *reachability graph* $\mathcal{RG}$ can be constructed that encodes all possible executions of the SPN. Reachability graph of the example SPN is shown in Fig. 2.2; its nodes are the seven reachable markings from the $\mathcal{RS}$ and its arcs represent possible transitions between the markings. In general, the reachability set, and thus the reachability graph, of an SPN can be infinite.

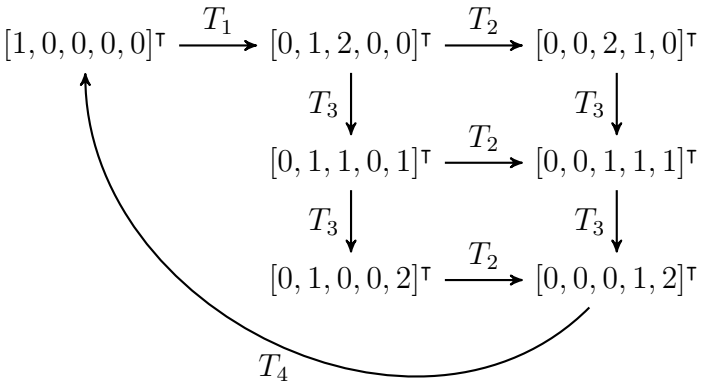


Figure 2.2: Reachability graph of the SPN in Fig. 2.1

The model and its behaviour that we have discussed so far, without taking into account function W from the definition of SPN, is a classic untimed Petri net. Function W defines timed behaviour of the model by defining marking-dependent transition firing rates. These rates are parameters of exponentially distributed random variables that define transition firing delays. We define the timed behaviour of SPNs in Section 2.1.3, after considering in the following sections some qualitative properties that SPNs share with the untimed PNs.

2.1.2 Some qualitative properties of stochastic Petri nets

SPNs inherit most qualitative properties of interest from the basic untimed Petri nets, which allows use of a rich set of methods for their qualitative analysis [9]. In this section we recall a few of the qualitative properties of SPNs that we make use of in the subsequent chapters.

Definition 2.1.4. *An SPN is said to be* live *if for every reachable marking* $\boldsymbol{m} \in \mathcal{RS}$ *and every transition* $T_i \in \mathcal{T}$ *there exists a sequence of transitions that, when fired from the marking* $\boldsymbol{m}$*, results in a marking in which transition* T_i *is enabled.*

In other words, liveness is the property that characterizes nets in which, no matter what sequence of transitions is fired from the initial marking, it is always possible to enable any transition of the SPN.

Definition 2.1.5. *An SPN is* bounded *if there exists a vector* $\boldsymbol{B} \in \mathbb{N}^{N_P}$ *which is an upper bound for the reachability set of the SPN, i.e.,* $\boldsymbol{m} \leq \boldsymbol{B}, \forall \boldsymbol{m} \in \mathcal{RS}$*, where* $\leq$ *denotes pointwise vector comparison.*

Obviously, an SPN is bounded if and only if the number of tokens in all places is bounded. Another characterization of the boundedness is based on cardinality of the reachability set: an SPN is bounded if and only if its reachability set is finite.

Definition 2.1.6. *An* S-invariant *or* place invariant *of an SPN is a nonzero vector* $\boldsymbol{U} \in \mathbb{N}^{N_P}$ *such that* $\boldsymbol{U}^\intercal \boldsymbol{m}_0 = \boldsymbol{U}^\intercal \boldsymbol{m}$ *for all reachable markings* $\boldsymbol{m} \in \mathcal{RS}$*.*

The product $\boldsymbol{U}^\intercal \boldsymbol{m}$ can be considered as a weighted sum of numbers of tokens in places for which the corresponding elements of $\boldsymbol{U}$ are nonzero; existence of an S-invariant thus signifies that the total weighted number of tokens in these places is conserved during execution of the SPN.

Definition 2.1.7. *The set of places for which corresponding elements of an S-invariant* $\boldsymbol{U}$ *are nonzero is called the* support *of the S-invariant* $\boldsymbol{U}$*. If an invariant* $\boldsymbol{U}$ *1) has a minimal support (in the sense that there is no S-invariant whose support is a proper subset of the support of* $\boldsymbol{U}$*), and 2)* $\boldsymbol{U}$ *is a minimal vector among the S-invariants (in the sense that there are no S-invariant* $\boldsymbol{U}' \leq \boldsymbol{U}$ *and an index* k *for which* $U'_k < U_k$*) it is called a* minimal support *S-invariant.*

For an arbitrary SPN the set of minimal support S-invariants is finite and forms a basis of all its S-invariants [32], in the sense that all S-invariants of an SPN can be represented as linear combinations of the minimal support S-invariants.

2.1.3 Timed behaviour of stochastic Petri nets

So far, timed behaviour was not considered. Firing delays of enabled transitions are exponentially distributed with marking-dependent rates defined by the function W. Via function W, various firing semantics can be defined. In this thesis we consider the following firing semantics:

- *single server (SS)* semantics, where the firing delay of an enabled transition T_i in marking $\boldsymbol{m}$ is exponentially distributed with rate that is independent of the enabling degree (we write $W(T_i, \boldsymbol{m}) = W(T_i)$, for all $\boldsymbol{m} \in \mathcal{RS}$); T_i can be considered to be processing only a single enabling set of tokens with the rate $W(T_i)$,

- *infinite server (IS)* semantics, where the firing delay of an enabled transition T_i in marking $\boldsymbol{m}$ is exponentially distributed with rate that depends linearly on the enabling degree (we write $W(T_i, \boldsymbol{m}) = e_i(\boldsymbol{m})W(T_i)$, for all $\boldsymbol{m} \in \mathcal{RS}$); in this case T_i can be considered to be processing in parallel each of the $e_i(\boldsymbol{m})$ enabling sets of tokens with rates equal to $W(T_i)$.

When a marking $\boldsymbol{m}$ is entered at epoch t, firing delays are sampled for all transitions enabled in $\boldsymbol{m}$ from exponential distributions with the rates defined as above, and the transition T_i with the smallest firing delay Δ_i fires next at epoch $t + \Delta_i$ and produces a new marking $\boldsymbol{m} - I(T_i) + O(T_i)$, at which point the firing delays are resampled again for all transitions that are enabled in the new marking. The non-determinism in untimed Petri nets is thus removed by the race policy among the exponential distributions corresponding to the enabled transitions.

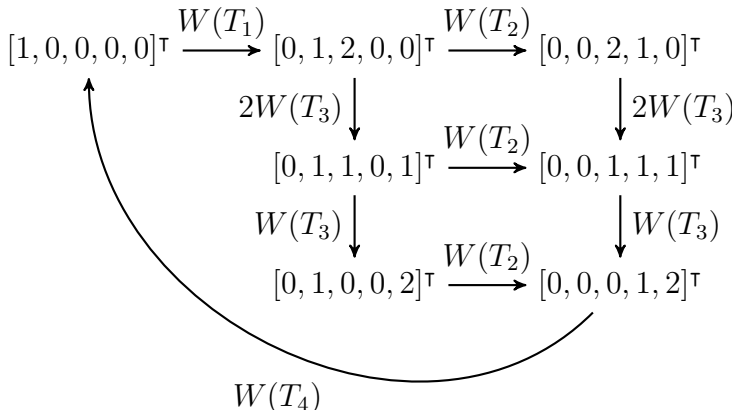


Figure 2.3: CTMC underlying the SPN in Fig. 2.1, assuming infinite server firing semantics.

From the above definition of the timed behaviour of an SPN model, it can be shown that the stochastic process underlying an SPN is a time-homogeneous continuous time Markov chain with a discrete state space that is equal to the reachability set $\mathcal{RS}$. Rates of the state transitions of the underlying CTMC depend on the firing rates and firing semantics of the SPN transitions and are defined as follows. For a marking $\boldsymbol{m} \in \mathcal{RS}$, let $\mathcal{E}(\boldsymbol{m}) = \{T_i : e_i(\boldsymbol{m}) > 0\}$ be the set of transitions that are enabled in marking $\boldsymbol{m}$. Then, in the underlying CTMC, there are state transitions from state $\boldsymbol{m}$ to states in the set $\{\boldsymbol{m} - I(T_i) + O(T_i) : T_i \in \mathcal{E}(\boldsymbol{m})\}$ where the rate of the exponentially distributed transition delay from state $\boldsymbol{m}$ to state $\boldsymbol{m} - I(T_i) + O(T_i)$ is $W(T_i, \boldsymbol{m})$. We denote this CTMC with $(\boldsymbol{m}(t))_{t \geq 0}$. Fig. 2.3 shows the CTMC underlying the example SPN, assuming infinite server firing semantics.

In this thesis we limit ourselves to the discussion of SPNs that are both bounded and live. It can be shown that the CTMC underlying an SPN that has these two properties is ergodic (we say also that the SPN is ergodic). Assuming some ordering of the markings from the reachability set, we denote the stationary probability

distribution of an ergodic SPN (i.e., of the CTMC $(\boldsymbol{m}(t))_{t\geq 0}$) with a row vector $\boldsymbol{\pi}$:

$$\pi_{\boldsymbol{m}_i} := \lim_{t\to\infty} Pr\{\boldsymbol{m}(t) = \boldsymbol{m}_i\}, \forall \boldsymbol{m}_i \in \mathcal{RS}. \tag{2.1}$$

Here on the left side we denote the element of $\boldsymbol{\pi}$ corresponding to a marking $\boldsymbol{m}_i$.

2.1.4 Stationary performance measures

Stationary performance measures of an SPN model, such as for example expected numbers of tokens in places or throughputs of transitions, can be calculated from the stationary marking distribution $\boldsymbol{\pi}$ of the SPN. We adopt a flexible SPN reward model for specification of performance measures [33]. In this reward model, reward rates are introduced for markings via function $r : \mathcal{RS} \to \mathbb{R}$ where $r(\boldsymbol{m})$ is a rate at which the reward accumulates while the process sojourns in marking $\boldsymbol{m} \in \mathcal{RS}$. Expected steady-state reward rate ρ can then be calculated in the following manner:

$$\rho = \sum_{\boldsymbol{m}\in\mathcal{RS}} r(\boldsymbol{m})\boldsymbol{\pi}_{\boldsymbol{m}} \tag{2.2}$$

In Table 2.1 we give specifications of some performance measures defined in terms of the reward function $r(\cdot)$. Descriptions of the performance measures that we consider are:

- Utilization of a transition is the probability that the transition is enabled.
- Throughput of a transition is the average number of firings of the transition in a unit of time.
- Average number of tokens in a place is the expected number of tokens in the place.
- Utilization of a place is the probability that the place is nonempty.
- Throughput of a place is the average number of tokens that are removed from the place in a unit of time.

Table 2.1: Definitions of common performance measures in terms of the reward function.

ρ	$r(\cdot)$
utilization $u(T_i)$ of transition T_i	$r(\boldsymbol{m}) = \delta_{\boldsymbol{m}\geq I(T_i)}$
throughput $x(T_i)$ of transition T_i	$r(\boldsymbol{m}) = \delta_{\boldsymbol{m}\geq I(T_i)} W(T_i, \boldsymbol{m})$
average number $n(P_i)$ of tokens in place P_i	$r(\boldsymbol{m}) = m_i$
utilization $u(P_i)$ of place P_i	$r(\boldsymbol{m}) = \delta_{m_i>0}$
throughput $x(P_i)$ of place P_i	$r(\boldsymbol{m}) = \sum_{T\in\mathcal{T}} \left(\delta_{\boldsymbol{m}\geq I(T)} W(T, \boldsymbol{m}) I_i(T)\right)$

2.2 Multi-valued decision diagrams

Multi-valued decision diagrams [14] are data structures used for encoding discrete functions of discrete variables. They are a generalisation of binary decision diagrams (BDD) which can encode Boolean functions of Boolean variables. BDDs have been used for digital circuit verification [34] and in model checking [35], while MDDs have been used in the performance analysis and verification of stochastic models [36], including GSPNs [37].

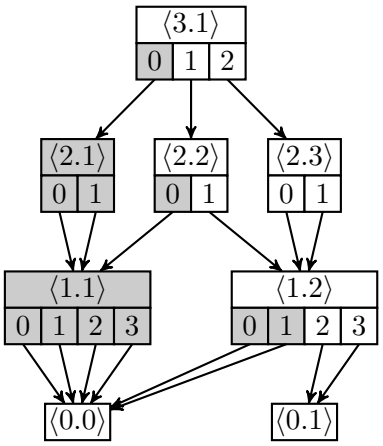
In particular, MDDs have been used to encode state spaces and transition functions of discrete event dynamic systems [15, 16], and efficient algorithms have been developed that can generate decision diagram encodings of very large state spaces [38, 39, 40] for many types of practical models. While efficiency of methods based on decision diagrams is in worst-case no better than using explicit representation of state space and state space generation based on traditional breadth-first search algorithm, in practice performance gains are significant for many models.

In this section we introduce MDDs and overview some of the usual operations that can be used for their creation and manipulation.

2.2.1 Definition

Definition 2.2.1. *Let* $K \in \mathbb{N}_{>0}$ *be a nonzero natural number, and let* $\mathcal{S}_k = \{0, 1, \ldots, n_k - 1\}, n_k \in \mathbb{N}_{>0}, k = 1, \ldots, K$ *and* $\mathcal{R} = \{0, 1, \ldots, n_0 - 1\}, n_0 \in \mathbb{N}_{>0}$ *be nonempty sets. A* multi-valued decision diagram *(MDD) over* $\hat{\mathcal{S}} := \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ *with values in* $\mathcal{R}$ *is a directed acyclic multi-graph with labeled arcs, and nodes organised into* $K + 1$ *levels. Nodes on levels* $K, \ldots, 1$ *are called* non-terminal nodes *while nodes on level* 0 *are called* terminal nodes*. A node in an MDD is denoted by* $\langle k.p \rangle$*, where* $k \in \{K, \ldots, 0\}$ *is the level of the node, and* p *is a unique node index in level* k*. For each non-terminal node* $\langle k.p \rangle$ *there are exactly* n_k *outgoing directed arcs labeled with* $0, 1, \ldots, n_k - 1$ *with the same source node* $\langle k.p \rangle$ *and destination nodes in level* $k - 1$*. These destination nodes are denoted by* $\langle k.p \rangle[0], \ldots, \langle k.p \rangle[n_k - 1]$*; for example* $\langle k.p \rangle[j] = \langle k - 1.q \rangle$ *means that there is an arc labeled with* j *from node* $\langle k.p \rangle$ *to node* $\langle k - 1.q \rangle$*. Readers familiar with MDDs will recognise that here MDDs are assumed to be* ordered *(i.e., arcs are allowed to point only to nodes in the neighbouring lower level). It is assumed that level* K *contains only a single root node* $\langle K.r \rangle$*, and that there are no* duplicate nodes*; two nodes* $\langle k.p \rangle$ *and* $\langle l.q \rangle$ *are duplicate if they are on the same level* $(k = l)$ *and all their corresponding arcs point to the same nodes* $(\langle k.p \rangle[j] = \langle l.q \rangle[j],\ \forall j \in \{0, 1, \ldots, n_k - 1\})$*. MDDs satisfying this requirement are said to be* quasi-reduced*. Level* 0 *contains exactly* n_0 *(terminal) nodes, denoted by* $\langle 0.0 \rangle, \ldots, \langle 0.n_0 - 1 \rangle$*, that have no outgoing arcs.*

Figure 2.4 contains an example of a decision diagram with 4 levels. An MDD A encodes a function $f_A : \hat{\mathcal{S}} \to \mathcal{R}$ in the following manner. To find the value $f_A(\boldsymbol{s})$, where $(s_k, \ldots, s_1) = \boldsymbol{s} \in \hat{\mathcal{S}} = \mathcal{S}_k \times \cdots \times \mathcal{S}_1$, one follows from the root of the MDD A the arcs labeled consecutively with $s_k, \ldots, s_1$ and reaches a terminal node, say $\langle 0.t \rangle$ for some index $t \in \mathcal{R}$. The index, in this case t, of the reached terminal node is the value of the function f_A in $\boldsymbol{s}$, $f_A(\boldsymbol{s}) = t$. The following definition introduces



$K = 3$

$\mathcal{S}_3 = \{0, 1, 2\}$
$\mathcal{S}_2 = \{0, 1\}$
$\mathcal{S}_1 = \{0, 1, 2, 3\}$

$\mathcal{R} = \{0, 1\}$

$\mathcal{S} = \{112, 113, 202, 203, 212, 213\}$

$\langle 3.1\rangle[1] = \langle 2.2\rangle$
$\langle 3.1\rangle[113] = \langle 2.2\rangle[13] = \langle 1.2\rangle[3] = \langle 0.1\rangle$
$\mathcal{B}(\langle 2.3\rangle) = \{02, 03, 12, 13\}$

Figure 2.4: Example of a decision diagram. Arc labels are indicated in the bottom halves of the decision diagram nodes. In an implementation, nodes with gray background do not need to be stored and remaining arcs whose labels have gray backgrounds can be redirected to point directly to node $\langle 0.0\rangle$.

a related notation with brackets for the procedure of following a sequence of arcs from an MDD node.

Definition 2.2.2. *For a node $\langle k.p\rangle$ in an MDD over $\hat{\mathcal{S}} := \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ and for a sequence of integers $\boldsymbol{s} = (s_k, \ldots, s_l) \in \mathcal{S}_k \times \cdots \times \mathcal{S}_l, K \geq k \geq l \geq 1$, node $\langle k.p\rangle[\boldsymbol{s}]$ is defined as the unique node that is reached from node $\langle k.p\rangle$ by following arcs labeled with elements from $\boldsymbol{s}$, and is given recursively with:*

$$\langle k.p\rangle[\boldsymbol{s}] = \begin{cases} \langle k.p\rangle[s_k] & \text{if } \boldsymbol{s} = (s_k) \text{ is a singleton,} \\ \langle k-1.q\rangle[\boldsymbol{t}] & \text{if } \boldsymbol{s} = (s_k)\boldsymbol{t} \text{ and } \langle k.p\rangle[s_k] = \langle k-1.q\rangle. \end{cases}$$

Here $(s_k)\boldsymbol{t}$ denotes the sequence obtained by concatenation of two sequences (s_k) and $\boldsymbol{t}$.

In this thesis, when an MDD is used to encode some set of states of a model, its levels $K, \ldots, 1$ correspond to submodels (places in case of SPNs), labels of arcs exiting level k correspond to local states of submodel k (markings of a place in case of SPNs) and the set $\mathcal{R}$ is set to $\{0, 1\}$. In this case we consider the MDD to encode the characteristic function of the encoded set of states, and we call sequences of the form $\boldsymbol{s} = (s_k, \ldots, s_l) \in \mathcal{S}_k \times \cdots \times \mathcal{S}_l, K \geq k \geq l \geq 1$ *substates* or, when $k = K$ and $l = 1$, *states*. In line with this, we introduce a notion of the set of substates encoded by a node of an MDD.

Definition 2.2.3. *For a node $\langle k.p\rangle$ in an MDD over $\hat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ with values in $\mathcal{R} = \{0, 1\}$, the* set $\mathcal{B}(\langle k.p\rangle)$ of substates encoded by the node $\langle k.p\rangle$ *is defined as*

$$\mathcal{B}(\langle k.p\rangle) = \{\boldsymbol{s} \in \mathcal{S}_k \times \cdots \times \mathcal{S}_1 : \langle k.p\rangle[\boldsymbol{s}] = \langle 0.1\rangle\}.$$

It is said that an MDD with root $\langle K.r\rangle$ encodes state space $\mathcal{S} \subseteq \hat{\mathcal{S}}$ *if the set of states encoded by the root node is equal to $\mathcal{S}$:*

$$\mathcal{B}(\langle K.r\rangle) = \mathcal{S}.$$

An important property of ordered quasi-reduced MDDs is that they are a canonical representation of the functions $\hat{\mathcal{S}} \to \mathcal{R}$; two functions $f, g : \hat{\mathcal{S}} \to \mathcal{R}$ are equal if and only if ordered quasi-reduced MDDs encoding f and g are isomorphic. A useful consequence of canonicity is that two non-terminal nodes $\langle k.p \rangle, \langle l.q \rangle$ of an MDD encode the same set if and only if $k = l$ and $p = q$, that is if they are the same node—otherwise, it can be shown that $\langle k.p \rangle$ and $\langle k.q \rangle$ are either duplicate nodes or that there exist duplicate nodes in one of the lower levels $k-1, \ldots, 1$, violating the quasi-reduced property from the definition of the MDD.

In the implementation of decision diagrams, nodes of the decision diagram from which all paths lead to node $\langle 0.0 \rangle$ do not need to be stored, since they can be deduced from the rest of the decision diagram. In the rest of the paper it is assumed that nodes with this property are omitted from the decision diagrams and that the remaining arcs that would point to such nodes point directly to node $\langle 0.0 \rangle$. If the MDD has values in $\mathcal{R} = \{0, 1\}$ and is used to encode the characteristic function of some set, then the following holds for all remaining non-terminal nodes:

$$\forall \langle k.p \rangle, \forall s_k \in \mathcal{S}_k, \quad \mathcal{B}(\langle k.p \rangle[s_k]) = \emptyset \iff \langle k.p \rangle[s_k] = \langle 0.0 \rangle.$$

Examples of these nodes and arcs are depicted in Figure 2.4 with gray backgrounds.

2.2.2 Operations with multi-valued decision diagrams

In this section we introduce some elementary operations on MDDs. For our purposes, we distinguish four types of MDDs, depending on the domain and range of the function that an MDD encodes. In the following, let $\hat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$ be a Cartesian product of $K > 0$ nonempty sets, as in the previous section.

- We call an MDD that encodes a function $f : \hat{\mathcal{S}} \to \{0, 1\}$ a *binary-terminal multi-valued decision diagram* (BTMDD). BTMDDs can be used to encode subsets of the set $\hat{\mathcal{S}}$. We use them to encode sets of states of SPNs.

- For $f : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \to \{0, 1\}$, the MDD is a *binary-terminal multi-valued matrix diagram* (BTMxD). BTMxDs can be used to encode relations on $\hat{\mathcal{S}}$. We use them to encode transition functions of SPNs.

- For $n_0 \in \mathbb{N}_{>0}$ and $f : \hat{\mathcal{S}} \to \{0, \ldots, n_0 - 1\}$ the decision diagram is a *multi-terminal multi-valued decision diagram* (MTMDD).

- For $n_0 \in \mathbb{N}_{>0}$ and $f : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \to \{0, \ldots, n_0 - 1\}$ the decision diagram is a *multi-terminal multi-valued matrix diagram* (MTMxD). We use MTMxDs in the generation of BTMxDs.

In the implementation of the algorithms from this thesis, we use a C++ programming library MEDDLY [41] that supports creation and manipulation of all of the above types of MDDs. In the rest of this section, we briefly introduce several basic operations on MDDs that are supported by this library and that can be used in order to create MDDs from scratch and to manipulate them.

Generating simple MDDs For a vector $\boldsymbol{s} = (s_K, \ldots, s_1) \in \hat{\mathcal{S}}$, and for a value $t \in \{0, \ldots, n_0 - 1\}$ an MTMDD (or BTMDD if $n_0 = 2$) can be generated that encodes a function $f : \hat{\mathcal{S}} \to \{0, \ldots, n_0 - 1\}$ which maps $\boldsymbol{s}$ to t and other vectors to 0; more precisely, the resulting function is defined with

$$f(\boldsymbol{x}) = \begin{cases} t, & \text{if } \boldsymbol{x} = \boldsymbol{s}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

Any number of the elements of $\boldsymbol{s}$ can be set to a special value DONT_CARE, meaning that the corresponding function arguments can have any allowed value and the generated function returns t if and only if the rest of the function arguments are equal to the corresponding elements of $\boldsymbol{s}$. For example, if we denote with $\mathcal{C}$ the set of all indices of elements that are set to DONT_CARE, $\mathcal{C} := \{k : s_k = \text{DONT\_CARE}\}$, then the generated MDD encodes a function defined with

$$f(\boldsymbol{x}) = \begin{cases} t, & \text{if } x_k = s_k, \forall k \notin \mathcal{C}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

Generating simple MxDs Analogously for matrix diagrams, given a vector $(\boldsymbol{s}, \boldsymbol{s}') = (s_K, \ldots, s_1, s'_K, \ldots, s'_1) \in \hat{\mathcal{S}} \times \hat{\mathcal{S}}$ and a value $t \in \{0, \ldots, n_0 - 1\}$, an MTMxD (or BTMxD if $n_0 = 2$) can be generated that encodes a function $f : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \to \{0, \ldots, n_0 - 1\}$ defined with

$$f(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} t, & \text{if } \boldsymbol{x} = \boldsymbol{s} \text{ and } \boldsymbol{x}' = \boldsymbol{s}', \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

In this case also any number of the elements of $(\boldsymbol{s}, \boldsymbol{s}')$ can be set to the value DONT_CARE with the same result as above. For example, if we denote with $\mathcal{C}$ and $\mathcal{C}'$ the sets of all indices of elements of $\boldsymbol{s}$ and $\boldsymbol{s}'$, respectively, that are set to DONT_CARE, $\mathcal{C} := \{k : s_k = \text{DONT\_CARE}\}$ and $\mathcal{C}' := \{k : s'_k = \text{DONT\_CARE}\}$, then the generated MxD encodes a function defined with

$$f(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} t, & \text{if } x_k = s_k, \forall k \notin \mathcal{C} \\ & \quad \text{and } x'_k = s'_k, \forall k \notin \mathcal{C}', \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

In addition, any number of elements of $\boldsymbol{s}$ (but not of $\boldsymbol{s}'$) can be set to a special value DONT_CHANGE. Denote the set of all elements of $\boldsymbol{s}$ that are set to DONT_CHANGE with $\mathcal{D} := \{k : s_k = \text{DONT\_CHANGE}\}$. In this case it is required that the elements of $\boldsymbol{x}$ with indices in $\mathcal{D}$ are equal to the corresponding elements of $\boldsymbol{x}'$ in order for the function to map to t, i.e., the generated MxD encodes the function defined with

$$f(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} t, & \text{if } x_k = s_k, \forall k \notin \mathcal{C} \cup \mathcal{D} \\ & \quad \text{and } x'_k = s'_k, \forall k \notin \mathcal{C}' \cup \mathcal{D} \\ & \quad \text{and } x_k = x'_k, \forall k \in \mathcal{D}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

Table 2.2: Operations (MTMDD A, MTMDD B) $\rightarrow$ MTMDD C and analogous for (MTMxD A, MTMxD B) $\rightarrow$ MTMxD C

Operation	Result
$C \leftarrow \text{PLUS}(A, B)$	$f_C(\boldsymbol{x}) = f_A(\boldsymbol{x}) + f_B(\boldsymbol{x})$
$C \leftarrow \text{MINUS}(A, B)$	$f_C(\boldsymbol{x}) = f_A(\boldsymbol{x}) - f_B(\boldsymbol{x})$
$C \leftarrow \text{MULTIPLY}(A, B)$	$f_C(\boldsymbol{x}) = f_A(\boldsymbol{x}) f_B(\boldsymbol{x})$
$C \leftarrow \text{DIVIDE}(A, B)$	$f_C(\boldsymbol{x}) = f_A(\boldsymbol{x}) / f_B(\boldsymbol{x})$ (integer division)
$C \leftarrow \text{MIN}(A, B)$	$f_C(\boldsymbol{x}) = \min(f_A(\boldsymbol{x}), f_B(\boldsymbol{x}))$
$C \leftarrow \text{MAX}(A, B)$	$f_C(\boldsymbol{x}) = \max(f_A(\boldsymbol{x}), f_B(\boldsymbol{x}))$

Generating projections Decision diagrams that encode projections to a particular variable can be created: for a given index k, an MTMDD (or BTMDD) can be created that encodes the function which maps $\boldsymbol{x} = (x_K, \ldots, x_1) \mapsto x_k$ for every $\boldsymbol{x} \in \hat{\mathcal{S}}$. Similarly, MTMxDs and BTMxDs can be created that encode projections to either unprimed or primed variables.

Operations on MDDs The above operations allow creation of decision diagrams that encode very simple functions and relations. To encode more complex functions and relations, one can combine these simple MDDs using a variety of operators. Tables 2.2 to 2.5 contain an overview of basic operators supported by MEDDLY, some of which we use in this thesis. For example, Table 2.2 lists operations that take two MTMDDs (or MTMxDs) A and B that encode functions f_A and f_B and return an MTMDD (or MTMxD, respectively) C that encodes function f_C as defined in the right column of the table. Operations BFS and DFS from table 2.5 can be used to generate a reachability set of a Petri net. They generate the reachability set by starting from a set of markings represented by MDD A and then repeatedly applying transition relation represented by MxD B. BFS uses a breadth-first search algorithm and DFS uses a more efficient saturation algorithm [42].

Notes on complexity Finally, we note that the size of an MDD (i.e., the number of its nodes) depends both on the function that it encodes, and on the ordering of its variables which defines the ordering of the levels of the MDD. The same holds for the computational complexity of the operations on MDDs. In the worst case, size of an MDD as well as the complexity of operations on MDDs are exponential in the number of the function variables. For many practical applications, however, MDDs are obtained that very efficiently encode the needed functions.

2.3 Summary

In this chapter we have introduced stochastic Petri nets and discussed some of their qualitative properties that we make use of in the following chapters and defined their timed behaviour as well as some common performance measures. Then we

Table 2.3: Operations (MTMDD A, MTMDD B) $\rightarrow$ BTMDD C and analogous for (MTMxD A, MTMxD B) $\rightarrow$ BTMxD C

Operation	Result
$C \leftarrow$ GREATER_THAN(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) > f_B(\boldsymbol{x})$
$C \leftarrow$ GREATER_EQUAL(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) \geq f_B(\boldsymbol{x})$
$C \leftarrow$ LESS_THAN(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) < f_B(\boldsymbol{x})$
$C \leftarrow$ LESS_EQUAL(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) \leq f_B(\boldsymbol{x})$
$C \leftarrow$ EQUAL(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) = f_B(\boldsymbol{x})$
$C \leftarrow$ NOT_EQUAL(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) \neq f_B(\boldsymbol{x})$

Table 2.4: Operations (BTMDD A, BTMDD B) $\rightarrow$ BTMDD C and analogous for (BTMxD A, BTMxD B) $\rightarrow$ BTMxD C

Operation	Result
$C \leftarrow A \cup B$	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) = 1$ or $f_B(\boldsymbol{x}) = 1$
$C \leftarrow A \cap B$	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) = 1$ and $f_B(\boldsymbol{x}) = 1$
$C \leftarrow A \setminus B$	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow f_A(\boldsymbol{x}) = 1$ and $f_B(\boldsymbol{x}) = 0$

Table 2.5: Operations (BTMDD A, BTMXD B) $\rightarrow$ BTMDD C

Operation	Result
$C \leftarrow$ PRE_IMAGE(A, B)	$f_C(\boldsymbol{x}) = 1 \Leftrightarrow \exists \boldsymbol{y}$ s.t. $f_A(\boldsymbol{y}) = 1$ and $f_B(\boldsymbol{x}, \boldsymbol{y}) = 1$
$C \leftarrow$ POST_IMAGE(A, B)	$f_C(\boldsymbol{y}) = 1 \Leftrightarrow \exists \boldsymbol{x}$ s.t. $f_A(\boldsymbol{x}) = 1$ and $f_B(\boldsymbol{x}, \boldsymbol{y}) = 1$
BFS(A, B)	fixpoint of $A \cup$ POST_IMAGE(A, B)
DFS(A, B)	fixpoint of $A \cup$ POST_IMAGE(A, B)

have defined multi-valued decision diagrams and discussed some of the operations that can be used for their creation and manipulation.

3

Perfect sampling in stochastic Petri nets

Simulation is widely used in stationary performance analysis of stochastic Petri nets with large state spaces, when exact numerical solution of the underlying Markov chain is infeasible. When simulation is used for stationary analysis, the *warm-up period* of a simulation run needs to be discarded. This period is the initial period of the simulation run in which state of the simulated system strongly depends on the initial state from which the simulation was started, and any estimation of performance indices in this period will be subject to this *initialisation bias*. After the warm-up period has elapsed, statistical methods can be used during the *sampling period* of the simulation run to estimate stationary performance indices of interest. However, the above procedure requires estimation of the length of the warm-up period, which can in some cases be done prior to simulation and in general can be performed during the simulation [43] [44]. Accuracy of the derived performance indices and their relevance—whether they pertain to the long-term behaviour of the system which is of interest in the stationary analysis or they are too biased by the initial state of the simulation—depend on the quality of the estimation of the warm-up period length.

An alternative to the above approach is to estimate the stationary performance indices by sampling directly from the stationary probability distribution of the model; this is referred to as *perfect* or *exact sampling* or *perfect simulation* [17]. These samples are perfect in the sense that they are distributed according to the exact stationary distribution of the model, and not according to the approximate stationary distribution which is obtained from sufficiently long simulation runs. If performance of a perfect sampling algorithm for a given model is good enough to obtain enough samples needed to achieve the precision required in estimation of the sought performance indices, then perfect sampling can be used instead of the standard simulation of the model. In the contrary, if performance of the algorithm prohibits the collection of the sufficient number of samples, a smaller number of samples can be obtained and used as initial states of the stationary simulation runs. Since such initial states are distributed according to the stationary probability distribution, simulation can proceed without the warm-up period by immediately starting the sampling period. This can be advantageous in case of models for which the estimation of the length of the warm-up period is difficult. When perfect samples are used in this manner, it is enough to obtain a single perfect sample per

simulation run and the performance of perfect sampling is thus less critical.

Perfect sampling is based on a classic perfect sampling algorithm, coupling from the past [17]. Previous approaches tailor the CFTP algorithm to specific classes of models, relying on their specific structure or properties in order to efficiently implement the algorithm. For example, such approaches are based on monotonicity properties of the model and coupling [17], envelope methods [18], bounding chains [20, 19] and existence of blocking states for a subclass of stochastic Petri nets [21]. Unfortunately, all of these approaches require some restrictions on the structure of the net.

In this chapter we present an algorithm for perfect sampling in stochastic Petri nets with finite state spaces which implements CFTP using multi-valued decision diagrams [14]. The algorithm exploits certain regularities present in the Markov chain underlying the SPN model to greatly speed up the CFTP algorithm. In contrast to previous work, the proposed algorithm is general and does not require any structural conditions on the SPNs. Although in the worst case the time and space complexities of the algorithm are more than exponential in the size of the SPN (because the state space must be generated), we observed that in many practical cases the structure of the model's underlying reachability set has regularities that can be efficiently exploited by decision diagrams and for which the performance of the algorithm is acceptable even in the case of very large state spaces.

We also present a tool called Stochastic Petri Nets Perfect Sampling (spnps) that implements the proposed algorithm and test its performance. The tool is implemented in C++ and uses an MDD library MEDDLY [41] for the creation and manipulation of the decision diagrams.

This chapter is structured as follows. First in Section 3.1 we recall the CFTP algorithm. Then in Section 3.2 we introduce the perfect sampling algorithm and establish its correctness after which in Section 3.3 we present the tool spnps that implements the presented algorithm and test its performance on several models. Finally, in Section 3.4 we test the performance of the tool on a model from the class of fork-join queueing networks.

3.1 Coupling from the past

Coupling from the past [17] is an algorithm for obtaining samples from the stationary probability distribution of ergodic discrete time Markov chains (DTMC) with finite state spaces. It is based on simulating the DTMC starting from all states until the simulations corresponding to different starting states couple into a single state.

In some cases, the simulation can be performed starting from a subset of states instead of all states, which can greatly increase efficiency of the algorithm. An approach [21] closely related to the one we propose applies coupling from the past to event graphs (Petri nets where each place is restricted to having only a single input and a single output transition), for which the authors show that it is possible to perform the simulation starting from a small number of initial states. In contrast, we do not require assumptions on the structure of the net and base our approach

on a brute force version of coupling from the past—simulating from all states of the Petri net. We use decision diagrams to efficiently encode and store subsets of the state space and state transition functions needed in the algorithm.

Let $(X_n)_{n\in\mathbb{N}}$ be an ergodic discrete time Markov chain with finite state space $\mathcal{S}$ and transition probabilities $p_{ij}, i, j \in \mathcal{S}$, $(U_{-n})_{n\in\mathbb{N}}$ a sequence of independent uniformly distributed on $[0,1]$ continuous random variables, and $\phi : \mathcal{S} \times [0,1] \to \mathcal{S}$ a simulation update rule which respects the transition probabilities of the DTMC $(X_n)_{n\in\mathbb{N}}$, i.e., for all states $i, j \in \mathcal{S}$ and U an uniformly distributed on $[0,1]$ continuous random variable the following holds:

$$Pr\{\phi(i,U) = j\} = p_{ij}. \tag{3.1}$$

For a subset $\mathcal{A} \subseteq \mathcal{S}$ of the state space we denote with $\phi(\mathcal{A}, U)$ the set $\{\phi(s,U) : s \in \mathcal{A}\}$ of images of states in $\mathcal{A}$.

Under these assumptions, Algorithm 1 (if it terminates) produces a sample from the stationary probability distribution of the Markov chain $(X_n)_{n\in\mathbb{N}}$ [17]. The inner loop of the algorithm simulates the Markov chain starting from all states for m iterations, and the outer loop repeats this process for increasing values of m until simulations from all states couple into a single state. This state is returned as the sample. The number of iterations m that produces coupling is highly dependent on the Markov chain and the update rule ϕ.

Algorithm 1: CFTP$(\mathcal{S}, \phi, U_0, U_{-1}, \ldots)$

Data: State space $\mathcal{S}$, update rule ϕ and uniform on $[0,1]$ i.i.d. random variables $U_0, U_{-1}, \ldots$
Result: Sample from the stationary distribution.

```
1 begin
2     m ← 1;
3     repeat
4         A ← S;
5         for i = −m + 1 to 0 do
6             A ← φ(A, U_i);
7         m ← 2m;
8     until |A| = 1;
9     return s ∈ A;
```

If the simulations that start from different states couple in a finite expected number of steps then the algorithm terminates with probability 1. In the next section we construct update rule ϕ for stochastic Petri nets using efficient representation with MDDs and we prove that it couples in finite expected number of steps the simulations starting from all markings of the reachability set of the SPN. In other words, we implement Algorithm 1 for SPNs and show that it terminates in finite expected time, producing a sample from the stationary probability distribution.

3.2 Algorithm for perfect sampling in stochastic Petri nets

To implement coupling from the past, we first obtain a discrete time Markov chain by uniformizing the continuous time Markov chain underlying the stochastic Petri net. Then we define the update rule ϕ that we use for coupling and encode it using decision diagrams. These steps are explained in the following subsections.

3.2.1 Uniformization

First we uniformize the CTMC underlying the SPN in order to obtain a DTMC that we use in the CFTP algorithm. We assume a live and bounded stochastic Petri net (and therefore ergodic) with transition firing rates that depend on the enabling degree of the transitions, i.e., we restrict the function $W : \mathcal{T} \times \mathbb{N}^{N_P} \to \mathbb{R}_{>0}$ from the definition of the SPN to have the form

$$W(T_i, \boldsymbol{m}) = r_i(e_i(\boldsymbol{m})), \forall T_i \in ,\forall \boldsymbol{m} \in \mathcal{RS}, \tag{3.2}$$

where $r_i : \mathbb{N} \to \mathbb{R}_{\geq 0}, i \in \{1, \ldots, N_T\}$ are functions that map enabling degrees to firing rates of transitions. For simplicity of exposition, we assume that for all transitions the firing rate is equal to 0 if the enabling degree is 0, and the firing rate is nonzero otherwise:

$$\forall T_i \in \mathcal{T}, r_i(k) = 0 \text{ if and only if } k = 0. \tag{3.3}$$

This type of rate dependency can represent the single server firing semantics—by setting $r_i(k) := W(T_i)$—and infinite server firing semantics—by setting $r_i(k) := kW(T_i)$—as special cases, where with $W(T_i)$ we denote a marking independent base firing rate of transition T_i.

Since coupling from the past works with discrete time Markov chains, we use uniformization to obtain a discrete time Markov chain with the same stationary probability distribution as the continuous time Markov chain underlying the SPN. In the following we describe the uniformization coefficient that we use. Let $E_1, \ldots, E_{N_T}$ be maximum enabling degrees of transitions:

$$E_i = \max\{e_i(\boldsymbol{m}) : \boldsymbol{m} \in \mathcal{RS}\}, i = 1, \ldots, N_T \tag{3.4}$$

and let $R_1, \ldots, R_{N_T}$ be maximum firing rates of transitions:

$$R_i = \max\{r_i(k) : 0 \leq k \leq E_i\}, i = 1, \ldots, N_T. \tag{3.5}$$

We use uniformization coefficient $\Lambda := \sum_{i=1}^{N_T} R_i$ equal to the sum of maximum transition rates. We choose this uniformization coefficient as a result of balancing two requirements. First, we want a uniformization coefficient for which the uniformization is as efficient as possible (i.e., we would like a small Λ). Second, we want to be able to define update rule ϕ that for a given state $\boldsymbol{m} \in \mathcal{RS}$ and a given sample from the random variable U fires a single Petri net transition $T_{i(U)}$ (if $T_{i(U)}$ is enabled

in $\boldsymbol{m}$), and we want $T_{i(U)}$ to depend only on the sample from the random variable U and not on the state $\boldsymbol{m}$ to which we apply ϕ. Then, when we apply ϕ to a set of markings $\mathcal{A} \subseteq \mathcal{RS}$, we will need to perform a very simple operation of adding $-I(T_{i(U)}) + O(T_{i(U)})$ to all markings in $\mathcal{A}$ in which $T_{i(U)}$ is enabled. We want this property in order to be able to efficiently encode the update rule ϕ using decision diagrams and also to simplify the proof that the algorithm terminates in finite expected time. The above definition of Λ satisfies the second requirement while being as efficient as possible. For single server semantics, this uniformization coefficient Λ is equal to the sum $\sum_{i=1}^{N_T} W(T_i)$ of transition rates, as in [21]. For infinite server semantics, Λ is equal to the sum $\sum_{i=1}^{N_T} E_i W(T_i)$ of base transition rates multiplied by maximum enabling degrees of transitions.

3.2.2 Update rule

In this section we define the update rule ϕ by splitting it into a number of partial update rules and defining which of the partial update rules should be applied for a given sample from a random variable in the perfect sampling algorithm. By exploiting special structure present in the CTMC underlying an SPN, we are able to define very simple partial update rules which can be easily implemented using MDDs. For each Petri net transition T_i and its possible enabling degree $k \in \{0, \ldots, E_i\}$, we define a partial update rule $\psi_i^k : \mathcal{RS} \to \mathcal{RS}$ in the following manner. For a set $\mathcal{A} \subseteq \mathcal{RS}$ of markings, ψ_i^k fires the transition T_i in states for which the firing rate of transition T_i is at least $r_i(k)$, and leaves the rest of the states unchanged:

$$\begin{aligned}\psi_i^k(\mathcal{A}) := \{\boldsymbol{m} - I(T_i) + O(T_i) : \boldsymbol{m} \in \mathcal{A}, r_i(k) \leq r_i(e_i(\boldsymbol{m}))\} \\ \cup \{\boldsymbol{m} : \boldsymbol{m} \in \mathcal{A}, r_i(k) > r_i(e_i(\boldsymbol{m}))\}.\end{aligned} \tag{3.6}$$

Finally, we define the update rule ϕ for the perfect sampling algorithm with $\phi(\boldsymbol{m}, U) := \psi_{i(U)}^{k(U)}(\boldsymbol{m})$, where $i(U)$ is defined as the unique transition index such that

$$\Lambda^{-1} \sum_{i=1}^{i(U)-1} R_i \leq U < \Lambda^{-1} \sum_{i=1}^{i(U)} R_i, \tag{3.7}$$

and $k(U)$ is an enabling degree associated with the minimum firing rate of transition $T_{i(U)}$ that is larger than $\Lambda U - \sum_{i=1}^{i(U)-1} R_i$:

$$k(U) \in \arg\min_k \left\{ r_{i(U)}(k) : r_{i(U)}(k) \geq \Lambda U - \sum_{i=1}^{i(U)-1} R_i \right\}. \tag{3.8}$$

Note that if for some transition T_i and two enabling degrees $k_1 \neq k_2$ transition rates are equal, $r_i(k_1) = r_i(k_2)$, then the partial update rules $\psi_i^{k_1}$ and $\psi_i^{k_2}$ will also be equal and $k(U)$ will not be uniquely determined. This happens, for instance, in case of single server firing semantics, where the firing rate of transition T_i is equal to $W(T_i)$ for any nonzero enabling degree. In this case we can simply take the smallest $k(U)$ and discard the duplicate partial update rules (i.e., we redefine $k(U)$ to return the minimum enabling degree from the set of enabling degrees in 3.8). Because of this, we can assume without loss of generality that for every transition T_i, rates

for different enabling degrees are different, and that the enabling degree $k(U)$ is unique. Further, this ensures that the probability of selecting partial update rule ψ_i^k in a step of perfect sampling algorithm is nonzero for all i and k.

With partial update rules ψ_i^k defined as above, we implement Algorithm 1 using multi-valued decision diagrams to encode state space $\mathcal{S}$ (which is in our case the reachability set $\mathcal{RS}$ of the SPN), set $\mathcal{A}$ and partial update rules ψ_i^k. At each step of the inner loop, for the associated random variable U we select a partial update rule $\psi_{i(U)}^{k(U)}$ as described above and apply it to the set $\mathcal{A}$. We describe encoding of the partial update rules and the resulting algorithm in the following section.

3.2.3 Encoding partial update rules using decision diagrams

We encode sets of Petri net markings and transition functions using MDDs in which each level corresponds to marking of a Petri net place. As mentioned before, efficiency of operations on decision diagrams depends on the ordering of the variables and finding the optimal ordering is computationally very expensive [45]. To avoid bias that could be introduced by selecting the ordering using a fixed heuristic, in our tests we select an efficient ordering of the Petri net places (and thus of the decision diagram variables) manually by a process of trial and error. In the following, we assume some ordering of the places and describe the construction of decision diagrams.

For $i = 1, \ldots, N_P$, let B_i be the bound on the number of tokens in place P_i. We assume the bounds B_i are known. In principle this is not a limitation of our approach, because for bounded Petri nets it is possible to generate an MDD encoding the reachability set without knowledge of the bounds [40] and then the bounds on the numbers of tokens in places can be obtained from this MDD. We define potential reachability set $\hat{\mathcal{S}}$ of the Petri net as the Cartesian product of possible markings for all places:

$$\hat{\mathcal{S}} = \prod_{i=1}^{N_P} \{0, \ldots, B_i\}. \tag{3.9}$$

To encode partial update rule ψ_i^k with a decision diagram, we first encode two functions for each transition T_i, $\mathrm{STEP}_i : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \to \{0, 1\}$ and $\mathrm{EDEG}_i : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \to \mathbb{N}$. Using Algorithm 2, we encode in STEP_i a characteristic function of a step relation on $\hat{\mathcal{S}}$ that corresponds to firing of the transition T_i, i.e., the function defined by

$$\mathrm{STEP}_i(\boldsymbol{m}, \boldsymbol{m}') = 1 \Leftrightarrow \boldsymbol{m} - I(T_i) + O(T_i) = \boldsymbol{m}'. \tag{3.10}$$

In Algorithm 2, we first store in STEP_i a characteristic function of a relation on $\hat{\mathcal{S}}$ such that two possible markings are in relation if they don't differ on places that are neither input nor output places of transition T_i. Then, for each place P_j that is input or output place of transition T_i we define a relation EQUAL(PROJ', DIFF) such that two possible markings are in this relation if they differ in place P_j by exactly $-I_j(T_i) + O_j(T_i)$. We intersect STEP_i with this relation. It is easy to see that the algorithm generates the characteristic function of a relation satisfying (3.10).

Next, using Algorithm 3, we encode in EDEG_i a function that returns the enabling degree of transition T_i in the marking represented by the first argument:

$$\mathrm{EDEG}_i(\boldsymbol{m}, \boldsymbol{m}') = e_i(\boldsymbol{m}). \tag{3.11}$$

Algorithm 2: ENCODE_STEP($SPN, \hat{\mathcal{S}}, i$)

Data: Stochastic Petri net SPN, potential reachability set $\hat{\mathcal{S}}$ and transition index i.
Result: Encoding of step function for transition T_i.

```
1 begin
2     STEP_i ← {(m, m', 1) : m, m' ∈ B, I_j(T_i) = 0 and O_j(T_i) = 0 ⇒ m_j = m'_j};
3     for j such that I_j(T_i) ≠ 0 or O_j(T_i) ≠ 0 do
4         PROJ' ← {(m, m', m'_j) : m, m' ∈ Ŝ};
5         DIFF ← {(m, m', m_j − I_j(T_i) + O_j(T_i)) : m, m' ∈ Ŝ};
6         STEP_i ← STEP_i ∩ EQUAL(PROJ', DIFF);
7     return STEP_i;
```

This algorithm is a straightforward calculation of the enabling degree of a transition, performed efficiently over the entire set of possible markings by the functions operating on MDDs.

Algorithm 3: ENCODE_EDEG($SPN, \hat{\mathcal{S}}, i$)

Data: Stochastic Petri net SPN, potential reachability set $\hat{\mathcal{S}}$ and transition index i.
Result: Encoding of enabling degree function for transition T_i.

```
1 begin
2     EDEG_i ← {(m, m', ∞) : m, m' ∈ Ŝ};
3     for j such that I_j(T_i) > 0 do
4         PROJ ← {(m, m', m_j) : m, m' ∈ Ŝ};
5         INP ← {(m, m', I_j(T_i)) : m, m' ∈ Ŝ};
6         EDEG_i ← MIN(EDEG_i, DIVIDE(PROJ, INP));
7     return EDEG_i;
```

Now we can finally encode partial step function ψ_i^k as a characteristic function of a relation on $\hat{\mathcal{S}}$, as shown in Algorithm 4. In this algorithm we encode ψ_i^k as a union of a step relation on markings for which the firing rate of transition T_i is larger than rate $r_i(k)$, and an identity relation on the rest of the markings. As decision diagrams STEP_i and EDEG_i could be used multiple times in invocations of Algorithm 4 with different values of k, in the actual implementation we generate these decision diagrams only once and cache them.

Having encoded the update rule, the only missing ingredient for the implementation of the CFTP algorithm is the reachability set of the SPN. After we generate the reachability set $\mathcal{RS}$ of the SPN and encode partial update functions ψ_i^k with MDDs as described above, we use coupling from the past to obtain a sample from the stationary distribution of the net, as shown in Algorithm 5.

Because the size of an MDD encoding the reachability set is in the worst case exponential in the size of the SPN, and the same holds in general for MDDs encoding subsets of the reachability set and functions representing partial update rules, performing a single iteration in line 15 of Algorithm 5 is in EXPSPACE, while the number of these iterations depends on stochastic properties of the SPN. Aside from

Algorithm 4: ENCODE_PSI($SPN, \hat{\mathcal{S}}, i, E_i, k$)

Data: Stochastic Petri net SPN, potential reachability set $\hat{\mathcal{S}}$, transition index i, maximum enabling degree E_i and enabling degree k.
Result: Encoding of partial update rule ψ_i^k.

1 **begin**
2 $\quad$ EDEG$_i \leftarrow$ ENCODE_EDEG($SPN, \hat{\mathcal{S}}, i$);
3 $\quad$ GEQ $\leftarrow \bigcup_{\substack{j \in \{0,1,\ldots,E_i\} \text{ s.t.} \\ r_i(j) \geq r_i(k)}}$ EQUAL(EDEG$_i$, $\{(\boldsymbol{m}, \boldsymbol{m}', j) : \boldsymbol{m}, \boldsymbol{m}' \in \hat{\mathcal{S}}\}$);
4 $\quad$ LT $\leftarrow \bigcup_{\substack{j \in \{0,1,\ldots,E_i\} \text{ s.t.} \\ r_i(j) < r_i(k)}}$ EQUAL(EDEG$_i$, $\{(\boldsymbol{m}, \boldsymbol{m}', j) : \boldsymbol{m}, \boldsymbol{m}' \in \hat{\mathcal{S}}\}$);
5 $\quad$ STEP$_i \leftarrow$ ENCODE_STEP($SPN, \hat{\mathcal{S}}, i$);
6 $\quad$ NOSTEP $\leftarrow \{(\boldsymbol{m}, \boldsymbol{m}, 1) : \boldsymbol{m} \in \hat{\mathcal{S}}\}$;
7 $\quad$ $\psi_i^k \leftarrow$ (STEP$_i \cap$ GEQ) $\cup$ (NOSTEP $\cap$ LT);
8 $\quad$ **return** ψ_i^k;

these iterations and the generation of the reachability set in line 5, which is also in EXPSPACE, the rest of the steps of the algorithm are of a lower computational complexity.

In the next section we show that the described algorithm terminates in finite expected time, producing a sample from the stationary distribution of the CTMC underlying the SPN.

3.2.4 Proof of coupling

In this section we prove the correctness of the proposed algorithm.

Proposition 3.2.1. Correctness of perfect sampling algorithm for SPNs *Consider a bounded and live stochastic Petri net with firing rate dependency as described in Section 3.2.1 and let ϕ be an update rule defined by partial update rules ψ_k^i and selection functions $i(U)$ and $k(U)$ as defined in Section 3.2.2.*

Coupling from the past, given in Algorithm 1, terminates in finite expected time and returns a sample from the stationary probability distribution of the SPN.

Proof. It is enough to show that for any two markings $\boldsymbol{m}, \boldsymbol{m}' \in \mathcal{RS}$ of the SPN there exists a finite coupling sequence of samples $U_0, U_{-1}, \ldots, U_{-m+1}$, for some $m \in \mathbb{N}$, with nonzero probability and such that the sequence of partial update rules $\psi_{i(U_{-m+1})}^{k(U_{-m+1})}, \psi_{i(U_{-m+2})}^{k(U_{-m+2})}, \ldots, \psi_{i(U_0)}^{k(U_0)}$, when applied to $\boldsymbol{m}$ and $\boldsymbol{m}'$, yields the same marking. From the existence of such coupling sequence, by applying Borel-Cantelli lemma [46] it follows that markings $\boldsymbol{m}, \boldsymbol{m}'$ couple in finite expected number of steps. From this, and the finiteness of the reachability set of the SPN it follows that the reachability set will also couple into a single state in a finite expected number of steps. This state is then a sample from the stationary distribution of the SPN [17]. In the following, we construct the finite coupling sequence of samples.

Algorithm 5: COUPLING_SPN($SPN, \hat{\mathcal{S}}, U_0, U_{-1}, \ldots$)

Data: Stochastic Petri net SPN, potential reachability set $\hat{\mathcal{S}}$, uniform on $[0,1]$ i.i.d. random variables $U_0, U_{-1}, \ldots$
Result: Sample from the stationary distribution.

begin
  for $i = 1$ *to* N_T **do**
    STEP$_i \leftarrow$ ENCODE_STEP($SPN, \hat{\mathcal{S}}, i$);
    EDEG$_i \leftarrow$ ENCODE_EDEG($SPN, \hat{\mathcal{S}}, i$);
  $\mathcal{RS} \leftarrow$ DFS($\{\boldsymbol{m}_0\}, \bigcup_{i=1}^{N_T}$ STEP$_i$);
  for $i = 1$ *to* N_T **do**
    Compute maximum enabling degree E_i;
    for $k = 1$ *to* E_i **do**
      $\psi_i^k \leftarrow$ ENCODE_PSI($SPN, \hat{\mathcal{S}}, i, E_i, k$);
  Compute selection functions $i(U)$ and $k(U)$;
  $m \leftarrow 1$;
  repeat
    $\mathcal{A} \leftarrow \mathcal{RS}$;
    for $j = -m+1$ *to* 0 **do**
      $\mathcal{A} \leftarrow$ POST_IMAGE($\mathcal{A}, \psi_{i(U_j)}^{k(U_j)}$);
    $m \leftarrow 2m$;
  until $|\mathcal{A}| = 1$;
  return $\boldsymbol{s} \in \mathcal{A}$;

Let $\boldsymbol{m}_1, \boldsymbol{m}_1' \in \mathcal{RS}$ be two different markings of the Petri net. Denote with $d_1 = d(\boldsymbol{m}_1, \boldsymbol{m}_1')$ the length of the shortest directed path in the reachability graph from $\boldsymbol{m}_1$ to $\boldsymbol{m}_1'$ and let $\sigma_1 = T_{i_1}, T_{i_2}, \ldots, T_{i_{d_1}}$ be the shortest sequence of transitions such that $\boldsymbol{m}_1 \xrightarrow{\sigma_1} \boldsymbol{m}_1'$ (if there are several shortest sequences, select one of them). Since the reachability graph of the Petri net is finite, the length d_1 of this sequence is bounded by the finite diameter of the reachability graph.

By construction of partial update rules, for the sequence of transitions σ_1 there exists a corresponding sequence of partial update rules $\tau_1 = \psi_{i_1}^{k_1}, \psi_{i_2}^{k_2}, \ldots, \psi_{i_{d_1}}^{k_{d_1}}$ such that:

$$\tau_1(\{\boldsymbol{m}_1\}) := (\psi_{i_{d_1}}^{k_{d_1}} \circ \cdots \circ \psi_{i_2}^{k_2} \circ \psi_{i_1}^{k_1})(\{\boldsymbol{m}_1\}) = \{\boldsymbol{m}_1'\}. \tag{3.12}$$

In general, every partial update rule ψ_i^k, when applied to a marking, either fires corresponding transition T_i, or leaves the marking unchanged. By construction, applying sequence τ_1 to $\boldsymbol{m}_1$ fires a corresponding transition for every partial update rule in τ_1. We say that τ_1 is fully fireable from marking $\boldsymbol{m}_1$. Applying sequence τ_1 to $\boldsymbol{m}_1$ yields marking $\boldsymbol{m}_1' = \boldsymbol{m}_1 + \boldsymbol{\delta}_1$ for some nonzero vector $\boldsymbol{\delta}_1 \in \mathbb{N}^{N_P}$. If τ_1 is fully fireable $n \in \mathbb{N}$ times from $\boldsymbol{m}_1$ (that is, sequence τ_1^n, obtained by concatenating n copies of τ_1, is fully fireable from $\boldsymbol{m}_1$), the resulting marking will be equal to $\boldsymbol{m}_1 + n\boldsymbol{\delta}_1$. Because the reachability set is finite, τ_1 is fully fireable only a finite number

of times from $\boldsymbol{m}_1$. Otherwise, we would obtain an infinite sequence $\{\boldsymbol{m}_1 + n\boldsymbol{\delta}_1\}_{n\in\mathbb{N}}$ of different markings in the finite reachability set $RS(\boldsymbol{m}_0)$ (a contradiction). Let $l_1 \in \mathbb{N}$ be the maximum number of times that τ_1 is fully fireable from $\boldsymbol{m}_1$.

We now observe what happens when τ_1 is applied l_1 times to markings $\boldsymbol{m}_1$ and $\boldsymbol{m}'_1$. We denote:

$$\{\boldsymbol{m}_2\} := \tau_1^{l_1}(\{\boldsymbol{m}_1\}) \text{ and } \{\boldsymbol{m}'_2\} := \tau_1^{l_1}(\{\boldsymbol{m}'_1\}). \tag{3.13}$$

Since τ_1 is fully fireable l_1 times from $\boldsymbol{m}_1$, we have that $\boldsymbol{m}_2$ is obtained by firing l_1 times the sequence of transitions σ_1 from the marking $\boldsymbol{m}_1$:

$$\boldsymbol{m}_1 \xrightarrow{\sigma_1^{l_1}} \boldsymbol{m}_2. \tag{3.14}$$

Since $\{\boldsymbol{m}'_2\} = \tau_1^{l_1}(\{\boldsymbol{m}'_1\}) = \tau_1^{l_1+1}(\{\boldsymbol{m}_1\}) = \tau_1(\{\boldsymbol{m}_2\})$, and τ_1 is *not* fully fireable from $\boldsymbol{m}_2$ we have that $\boldsymbol{m}'_2$ is obtained by firing from $\boldsymbol{m}_2$ some strict subsequence σ'_1 of the sequence of transitions σ_1:

$$\boldsymbol{m}'_1 \xrightarrow{\sigma_1^{l_1-1}} \boldsymbol{m}_2 \xrightarrow{\sigma'_1} \boldsymbol{m}'_2, \text{ where } \sigma'_1 \subsetneqq \sigma_1. \tag{3.15}$$

Therefore, distance in the reachability graph between $\boldsymbol{m}_2$ and $\boldsymbol{m}'_2$ is strictly less than d_1, the length of sequence σ_1. To recapitulate, we have shown the following:

$$\begin{gathered}\exists l_1 \in \mathbb{N},\ \exists \boldsymbol{m}_2, \boldsymbol{m}'_2 \in \mathcal{RS} \text{ such that} \\ \tau_1^{l_1}(\{\boldsymbol{m}_1\}) = \{\boldsymbol{m}_2\},\ \ \tau_1^{l_1}(\{\boldsymbol{m}'_1\}) = \{\boldsymbol{m}'_2\} \text{ and} \\ d_2 := d(\boldsymbol{m}_2, \boldsymbol{m}'_2) < d_1.\end{gathered} \tag{3.16}$$

Repeating the above argument for markings $\boldsymbol{m}_2, \boldsymbol{m}'_2$, we obtain another sequence $\tau_2^{l_2}$ of partial update rules and markings $\boldsymbol{m}_3, \boldsymbol{m}'_3$ such that $d_3 := d(\boldsymbol{m}_3, \boldsymbol{m}'_3) < d_2$. Continuing in the same manner, for some $t \in \mathbb{N}$ we obtain markings $\boldsymbol{m}_t, \boldsymbol{m}'_t$ such that $d(\boldsymbol{m}_t, \boldsymbol{m}'_t) = 0$ or, equivalently, $\boldsymbol{m}_t = \boldsymbol{m}'_t$. Concatenating all obtained sequences of partial update rules we obtain a sequence $\tau = \tau_1^{l_1}\tau_2^{l_2}\dots\tau_t^{l_t}$ of partial update rules such that:

$$\tau_t^{l_t} \circ \dots \circ \tau_2^{l_2} \circ \tau_1^{l_1}(\{\boldsymbol{m}_1\}) = \tau_t^{l_t} \circ \dots \circ \tau_2^{l_2} \circ \tau_1^{l_1}(\{\boldsymbol{m}'_1\}). \tag{3.17}$$

By construction of selection functions $i(U)$ and $k(U)$, it is easy to see that there exists a finite sequence of samples corresponding to sequence τ and with nonzero probability. This finishes the proof. □

3.3 Tool spnps

In this section we present spnps, a tool for perfect sampling in stochastic Petri nets that implements the algorithm described in the previous section. The tool is implemented in C++ and is based on encoding in the form of multi-valued decision diagrams of the state space and transition functions of the DTMC obtained by uniformization of the CTMC underlying the SPN, in order to efficiently implement

the CFTP algorithm, as described previously. The tool can be obtained at the web page of one of its authors[1].

In addition to obtaining perfect samples, the tool can also be used in the analysis of models that exhibit multimodal behaviour. State spaces of such models can be partitioned into subsets (we also call them *modes*) between which there is very little communication (i.e., very low probability of transitioning from one subset to another subset). When simulating such models, the model's state usually stays confined to one of the subsets for a large number of simulation steps, resulting in meta-stable behaviour which can be easily mistaken for the stationary behaviour. Such multimodal behaviour can be detected by the behaviour of the tool during execution.

3.3.1 Objectives

This section describes main objectives of the tool, intended purpose and targeted users.

Table 3.1: Objectives of spnps

Application domain	Stationary performance analysis of SPN models
Targeted users	Researchers and analysts
Primary purpose	Sampling from stationary distribution of the SPN model
Secondary purpose	Detection of multimodal model behaviour

Primary purpose of the tool is obtaining samples from stationary probability distributions of stochastic Petri nets with finite state spaces. Secondary purpose of the tool is detection of multimodality of behaviour in SPN models with finite state spaces.

Intended use of the tool is as an aid in the stationary performance analysis of SPN models with finite state spaces. As explained in the introduction to this chapter, the obtained samples can be used directly to estimate stationary performance indices of interest, or they can be used as initial states of stationary simulation runs in place of a warm-up period.

The tool is intended to be used by researchers and analysts that employ stochastic Petri nets in stationary analysis of systems. It could be useful in case of problems in the simulation of the models indicating strong initialisation bias or multimodality of behaviour; these problems include strong sensitivity of the obtained performance indices to the initial state of the simulation, unreliable convergence of the stationary performance indices during the simulation, or simulation not capturing the expected range of model behaviour.

3.3.2 Functionality

Spnps is a command line application that allows the user to load from a file an SPN model together with place marking bounds and an ordering of places, and

[1] Ivan Stojic - Software, `http://www.dais.unive.it/~stojic/soft.html`

Table 3.2: Command line switches

Switch	Parameter type	Description
`-x` or `--xml`	`filename`	Input file
`-r` or `--rng`	`code`	PRNG
`-s` or `--seed`	`number`	PRNG seed
`-n` or `--num`	`number`	Number of samples
`-c` or `--card`	`number`	Target cardinality
`-v` or `--verbose`		Verbose output
`-d` or `--diagnostics`		Output of diagnostic information
`-t` or `--time`		Output of timing information

generate a chosen number of perfect samples from the reachability set of the SPN. When generating more than one sample, time taken to generate the reachability set $\mathcal{RS}$ and the simulation update rule ϕ in Algorithm 5 is ammortized by reusing these MDDs in the obtaining of further samples. Stopping criterion can be changed so that the perfect sampling algorithm is stopped before the completion, when the cardinality of the set $\mathcal{A}$ in Algorithm 5 becomes smaller than a chosen value instead of being equal to 1 as in the basic version of the algorithm. The entire set $\mathcal{A}$ is then returned as the result. Performing such incomplete perfect sampling runs can be useful in the analysis of models that exhibit multimodal behaviour. This is further detailed in Section 3.3.4.

Program Options

Table 3.2 lists command line switches supported by the tool. Switch `-x` selects an input file containing a description of an SPN model and is the only mandatory switch. Format and contents of the input file are described in the next subsection. Switch `-r` allows selection of quality level of a high quality pseudorandom number generator [47] (PRNG) and `-s` allows selection of PRNG seed; if the seed is omitted or set to 0, local time is used as the seed. Switch `-n` specifies the number of sampling runs that are to be performed by the tool. Switch `-c` sets the stopping cardinality—as explained before, when stopping cardinality is set to 1, samples from the stationary probability distribution are obtained, and larger values can be used to examine multimodal models. Finally, switch `-v` enables verbose output, and switches `-d` and `-t` enable output of additional information that is useful in debugging a model and testing the tool.

Input

Input file is assumed to be in the Petri Net Markup Language (PNML) [48] format. PNML is an XML-based syntax for Petri nets which aims at becoming the standard interchange format for Petri net tools. Since PNML is very flexible and extendable, leaving many format details to be defined based on specific needs of a tool using the format, we include in the tool distribution archive an XML Schema defining the particular format of PNML files that is supported by spnps. Fig. 3.1 shows part of

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="spnps01.xsd">
    <net id="Example" type="P/T net">
        <place id="P1">
            <initialMarking>
                <value>100</value>
            </initialMarking>
            <ddOrder>
                <value>0</value>
            </ddOrder>
            <bound>
                <value>100</value>
            </bound>
        </place>
        <!-- Place P2 omitted... -->

        <transition id="T1">
            <rate>
                <value>1.0</value>
            </rate>
            <timed>
                <value>true</value>
            </timed>
            <infiniteServer>
                <value>false</value>
            </infiniteServer>
        </transition>
        <!-- Transition T2 omitted... -->

        <arc id="P1 to T1" source="P1" target="T1">
            <inscription>
                <value>10</value>
            </inscription>
        </arc>
        <!-- Three arcs omitted... -->
    </net>
</pnml>
```

Figure 3.1: Example input file in PNML format; parts are omitted for brevity

the input file for the example SPN shown in Fig. 3.2. In the latter figure, all data that are expected in the input file are depicted: aside from the structure and initial marking of the SPN, additional data are firing rates and semantics for transitions and marking bounds and ordering of places in the generated MDDs. PNML files can contain multiple SPNs; in this case the tool loads the first net from the file.

Output

In normal usage, only the results of the sampling runs—perfect samples or sets of markings, depending on the stopping cardinality—are output to the standard output stream of the tool process. Using the verbose output switch `-v` causes additional output to be produced during the sampling procedure. Switch `-d` enables output of program options and loaded model data, and switch `-t` enables output of timing information. When the tool is invoked without any parameters, a message explaining the use of the tool is output. An example of the tool invocation and the output are shown in Fig. 3.3. The input file used in this example contains the small example SPN depicted in Fig. 3.2. Spnps is first invoked without any arguments and it outputs the usage information. Then it is invoked with input file set to

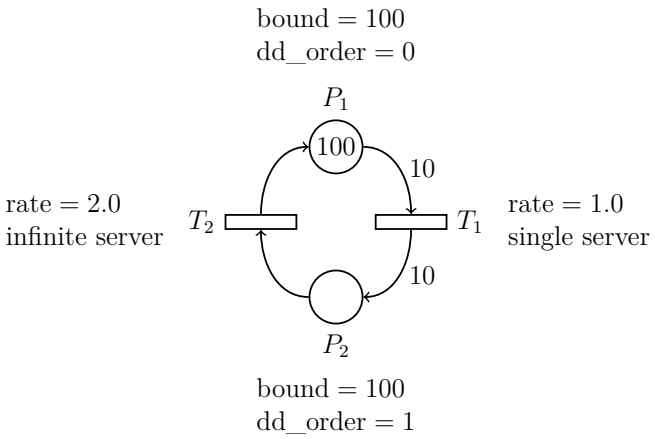


Figure 3.2: Example stochastic Petri Net with additional data expected in the input file, indicated next to corresponding places and transitions

Table 3.3: Interface of class `rng`

```
rng(int code, unsigned int seed, int verbose = 0)
double next()
```

`example.pnml` and with all output switches activated. The tool outputs program options and PRNG seed, some parsing diagnostics, data loaded from the input file, cardinality of the reachability set, progress of the sampling run containing the numbers of simulation steps and cardinalities of the final obtained sets and finally in the last line the obtained sample, in this case marking $[96, 4]^\intercal$.

3.3.3 Architecture

Spnps is a command line program written in C++. Aside from two helper functions that parse command line arguments and print usage information, the code is organised into three classes.

Class `rng`. This is a simple front-end to high quality pseudorandom number generators [47] which are available from the web page[2] of its authors. The interface of the class contains only two functions with signatures depicted in Table 3.3; constructor `rng` whose argument `code` specifies the PRNG quality level and argument `seed` specifies PRNG seed and function `next` which returns a pseudorandom floating point number between 0 and 1.

Class `spn`. This class represents a stochastic Petri net with additional data, as described in Section 3.3.2. The constructor `spn` loads the SPN and data from a file

[2]Software developed by the Canada Research Chair in Stochastic Simulation and Optimization, `http://simul.iro.umontreal.ca`

```
$ ./spnps
spnps version 0.1
Usage: ./spnps -x xml_file [-r code] [-s seed] [-n num] [-c card] [-v] [-d] [-t]
  -x or --xml input file
  -r or --rng (optional, code is one of 512, 1024, 19937, 44497, default = 1024)
              pseudorandom number generator
  -s or --seed (optional, seed >= 0, default = 0) PRNG seed, 0 specifies clock
  -n or --num (optional, default = 1) number of samples
  -c or --card (optional, default = 1) stopping cardinality
  -v or --verbose (optional) verbose output
  -d or --diagnostics (optional) output of diagnostic information
  -t or --time (optional) output of timing information

$ ./spnps -x example.pnml -v -d -t
Program options:
    XML file: 'example.pnml'
    RNG: WELL1024a
    RNG seed: use clock
    Samples: 1
    Stopping cardinality: 1
    Verbose output: 1
    Print diagnostic information: 1
    Print timing information: 1
RNG WELL1024a initialized with seed 1458864504.
Loading SPN from file 'example.pnml'.
    Using net with id="Example".
    Found 2 places.
    Found 2 transitions.
    Found 4 arcs.
    Loading done.
Loaded data:
    n = 2
    m = 2
    I =
        10 0
        0 1
    O =
        0 1
        10 0
    W = 1.0000000000000000e+00 2.0000000000000000e+00
    M0 = 100 0
    S = SS IS
    B = 100 100
    V = 0 1
Generating reachability set... done.
    cardinality = 101
Generating simulation update rule... done.
Init time = 4.4442700000000002e-01 s
Sampling (1 of 1):
    m = 1, card = 100
    m = 2, card = 99
    m = 4, card = 97
    m = 8, card = 93
    m = 16, card = 85
    m = 32, card = 69
    m = 64, card = 52
    m = 128, card = 28
    m = 256, card = 11
    m = 512, card = 2
    m = 1024, card = 1
    Sample time = 5.4229999999999999e-03 s
    Sample iterations = 1024
96 4
```

Figure 3.3: Example invocation and output of spnps

Table 3.4: Interface of class `spn`

`spn(char *xmlFile, int verbose = 0)`
various getter functions such as `int n()` and `const int *B()`

Table 3.5: Interface of class `sampler`

`sampler(spn &spn, rng &gen, int verbose = 0, int timing = 0)`
`set<vector<int> > sample(int card)`

in PNML format. Parsing of the input is implemented using library `libxml2`[3]. The loaded data can then be queried using several getter functions. Two example getter functions are listed in the second row of Table 3.4; they return number of places n and the vector of place marking bounds, respectively.

Class `sampler`. This class implements the perfect sampling algorithm. It uses library MEDDLY[4] [41] that supports creation and manipulation of several types of decision diagrams. Constructor `sampler` takes as parameters objects of type `spn` and `rng` defining the SPN and PRNG to be used in sampling, and generates MDD encoding the reachability set of the SPN and MDDs that encode the simulation update rule. Function `sample` can then be used to obtain a sample, by setting its parameter `card` to 1, or to perform an incomplete sampling run and obtain a set of markings, by setting `card` to a number greater than 1.

3.3.4 Use Cases

Two use cases are envisioned for the tool: sampling from the stationary distribution of an SPN and analysing multimodal model behaviour by performing incomplete sampling runs. The former use case, sampling, has been described in previous sections and the typical tool use for this purpose was shown in Fig. 3.3. For this use case, performance of the tool suggests that its usefulness is more likely to be in the obtaining of samples to use as initial states of simulation runs, and less likely in the estimation of performance indices by directly obtaining a large number of samples from the stationary probability distribution of the model. In the rest of this section the latter use case, analysing multimodal behaviour, is illustrated with a small example.

Analysing Multimodal Behaviour

Consider the SPN in Fig. 3.4, and assume that all transitions have equal firing rates and infinite server firing semantics. Transition T_3 is enabled only in marking $[0, M, 0, 0]^\intercal$, when all tokens are in place P_2, and its enabling degree is 1. Note that it may take a large number of steps for the SPN to reach this marking because the transitions T_1 and T_2 have infinite server firing semantics and will therefore result

[3]The XML C parser and toolkit of Gnome, `http://www.xmlsoft.org`

[4]Multi-terminal and Edge-valued Decision Diagram LibrarY, `http://meddly.sourceforge.net`

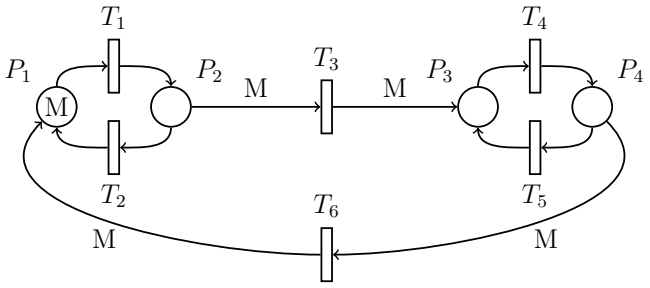


Figure 3.4: A bimodal SPN

in the behaviour tending to equalise the number of tokens in places P_1 and P_2. In the same marking $[0, M, 0, 0]^\intercal$, transition T_2 is enabled with enabling degree M. Since we assume equal firing rates and infinite server firing semantics, firing rate of transition T_2 in this marking is M times larger than the firing rate of transition T_3. Therefore, in the race between these two transitions, T_3 fires with probability $1/(M+1)$ and T_2 fires with probability $M/(M+1)$. For a large M, it is easy to see that in the simulation of this SPN, a large number of iterations will elapse between the enablings of transition T_3 and for each of these enablings the probability for transition T_3 to fire and transfer tokens to place P_3 will be small. Because of symmetry, the same holds for transition T_6. This is an example of a bimodal SPN, where the state space can be decomposed into two sets, $\{[M-i, i, 0, 0]^\intercal : 0 \leq i \leq M\}$ and $\{[0, 0, M-i, i]^\intercal : 0 \leq i \leq M\}$, between which there is very little communication.

If simulation is started in the marking depicted in Fig. 3.4, it will likely take a large number of iterations for the transition T_3 to fire. During this time, the SPN will show meta-stable behaviour, estimates of performance measures will converge, and the simulation will likely be stopped before the transition T_3 fires. This scenario is very likely if the analyst is not aware of the bimodality of the model behaviour. While in the case of this small example SPN multimodality can be easily deduced from the model description, in case of larger and more complex models it could easily go unnoticed.

Spnps can be used in the detection and analysis of multimodal models. For the example bimodal SPN with $M = 100$, running spnps with the verbose output enabled produces output shown in Fig. 3.5, left. The code `^C` at the end of the output signifies that the user has interrupted execution because it stopped making progress—cardinality of the obtained sets remained at 2 for increasing numbers of simulation steps m. This kind of behaviour of the perfect sampling algorithm shows that a standard simulation would need a very long warm-up period and is an indication of possible multimodality of the model behaviour. The user can further analyse the model behaviour by running the tool with target cardinality set to 2. Performing five incomplete sampling runs with this target cardinality produces the output shown in Fig. 3.5, right. For each of the incomplete sampling runs, the tool outputs the cardinality of the obtained set (here equal to 2 in all cases), and the markings in the obtained set. For example, the first three lines of output

```
$ ./spnps -x bimodal.pnml -v
[... some output omitted ...]
Sampling (1 of 1):
    m = 1, card = 201
    m = 2, card = 200
    m = 4, card = 198
    m = 8, card = 194
    m = 16, card = 187
    m = 32, card = 174
    m = 64, card = 146
    m = 128, card = 106
    m = 256, card = 60
    m = 512, card = 16
    m = 1024, card = 3
    m = 2048, card = 2
    m = 4096, card = 2
    m = 8192, card = 2
    m = 16384, card = 2
    m = 32768, card = 2
    m = 65536, card = 2
    m = 131072, card = 2
[... some output omitted ...]
    m = 67108864, card = 2
    m = 134217728, card = 2
    m = 268435456, card = 2
^C
```

```
$ ./spnps -x bimodal.pnml -n 5 -c 2
2
0 0 43 57
49 51 0 0
2
0 0 45 55
55 45 0 0
2
0 0 57 43
45 55 0 0
2
0 0 62 38
52 48 0 0
2
0 0 44 56
40 60 0 0
```

Figure 3.5: Left: perfect sampling run for the bimodal SPN with $M = 100$, interrupted by the user; right: incomplete sampling runs for the same model

are to be interpreted as a set $\{[0, 0, 43, 57]^\intercal, [49, 51, 0, 0]^\intercal\}$ containing two markings. From the output, the bimodal nature of the model is apparent: the two weakly communicating subsets of the reachability set have both completely coupled into single states. For more complex models, a further analysis of the obtained sets of markings may need to be performed to analyse the multimodality of the model behaviour.

3.3.5 Experiments

In this section we test the performance of spnps by running experiments on several models. Table 3.6 lists the models that were used for testing, along with short descriptions. Full descriptions and figures of some of the models can be found at the end of this section. For each combination of a model and parameters, 10 sampling runs have been performed. Rates of transitions were selected uniformly from the segment $[1, 10]$ for each testing run for all models. All tests were performed on a Linux system with a 2.40GHz Intel Xeon E5-2665 CPU.

Results of tests are reported in Table 3.7. First column *Model* specifies the tested model and second column *Sem* specifies the firing semantics that are used for all transitions of the model. N_P is number of places, N_T is number of transitions of the net, and $|\mathcal{RS}|$ is the size of the reachability set. Last two columns are measurements of the performance of the tool, taken as averages over 10 runs. *Init* is total execution time prior to the coupling phase of the algorithm, and includes the generation of decision diagrams that encode the reachability set and partial update rules. *Coupling* is the time needed for execution of coupling from the past.

Table 3.6: Tested models.

Model	Description
phil N	N dining philosophers (take both forks at once)
rphil N	N dining philosophers (take one fork at a time)
slot N	slotted protocol model with N nodes
contention N M	N CPUs, M tasks per CPU compete for a resource
loop N M	simple loop of N places with M tokens

Table 3.7: Test results.

Model	Sem	N_P	N_T	$\lvert\mathcal{RS}\rvert$	Init (s)	Coupling (s)
phil 10	SS	40	30	2.32×10^{4}	0.011	0.011
phil 20	SS	80	60	5.37×10^{8}	0.037	0.078
phil 50	SS	200	150	6.67×10^{21}	0.227	0.744
phil 100	SS	400	300	4.46×10^{43}	1.414	4.890
phil 200	SS	800	600	1.98×10^{87}	7.868	24.734
phil 500	SS	2000	1500	1.76×10^{218}	64.495	199.472
rphil 10	SS	60	70	4.68×10^{6}	0.030	0.077
rphil 20	SS	120	140	2.19×10^{13}	0.122	0.537
rphil 50	SS	300	350	2.25×10^{33}	0.841	6.598
rphil 100	SS	600	700	5.08×10^{66}	5.615	31.023
rphil 200	SS	1200	1400	2.58×10^{133}	26.613	165.598
slot 5	SS	50	50	1.72×10^{6}	0.015	1.562
slot 10	SS	100	100	8.49×10^{12}	0.072	113.182
contention 10 10	IS	31	30	2.62×10^{11}	0.083	0.136
contention 20 10	IS	61	60	1.29×10^{22}	0.345	1.342
contention 50 10	IS	151	150	5.45×10^{53}	2.417	23.282
contention 100 10	IS	301	300	1.27×10^{106}	10.901	268.370
loop 10 10	SS	10	10	9.24×10^{4}	0.005	0.039
loop 10 20	SS	10	10	1.00×10^{7}	0.019	0.253
loop 10 50	SS	10	10	1.26×10^{10}	0.169	4.904
loop 10 100	SS	10	10	4.26×10^{12}	0.995	30.004
loop 100 5	SS	100	100	9.20×10^{7}	0.113	20.286
loop 100 10	SS	100	100	4.26×10^{13}	0.208	65.187
loop 10 10	IS	10	10	9.24×10^{4}	0.024	0.261
loop 10 20	IS	10	10	1.00×10^{7}	0.114	4.242
loop 10 50	IS	10	10	1.26×10^{10}	1.267	275.773
loop 100 5	IS	100	100	9.20×10^{7}	0.544	278.475

The tool is very efficient for models **phil**, **rphil** and **contention**, allowing sam-

pling despite very large reachability sets. Performance of the algorithm on these models is comparable when one takes into account size of the reachability set and number of iterations needed for coupling. We note that these models are somewhat similar, as they are all composed of a number of small components that share resources.

For models **slot** and **loop** performance of the tool is worse. In case of model **slot** this is mainly due to explosion of decision diagrams during the execution of the algorithm. Investigation has shown that during execution of the coupling phase on model **slot**, peak number of nodes of decision diagram encoding set $\mathcal{A}$ is up to 100 times higher than when the algorithm executes on models **phil**, **rphil** and **contention** with comparable sizes of the reachability sets. This decreased efficiency of MDDs encoding subsets of the reachability set of model **slot** is likely due to this model being more complex than the above models.

Similarly, the MDD encoding is also less efficient for model **loop**. While performance of coupling is somewhat better than for model **slot**, this is mainly due to a smaller number of iterations. For model **loop** 100 with single server semantics performance of the algorithm drops in comparison to model **loop** 10 mainly due to the increased number of iterations.

Finally, we compare performance for single server semantics and infinite server semantics for the model **loop**. Figures 3.6 and 3.7 show number of states in set $\mathcal{A}$ and number of nodes in decision diagram encoding of set $\mathcal{A}$ during execution of the algorithm. We see that for the model with infinite server semantics more iterations are needed to couple all states, and that the number of nodes in the decision diagram is much higher than in the case of single server semantics, resulting in longer execution time of the algorithm.

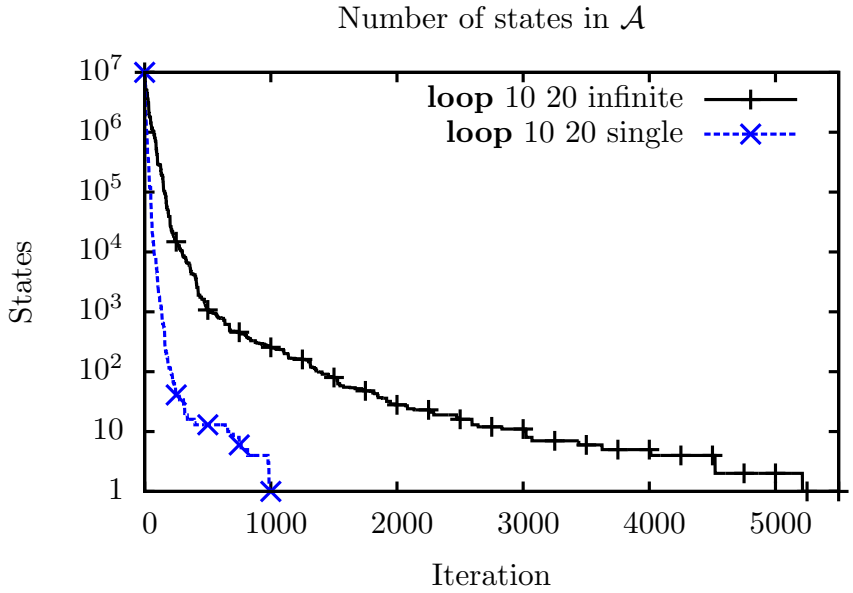


Figure 3.6: Number of states in set $\mathcal{A}$ during execution of coupling phase.

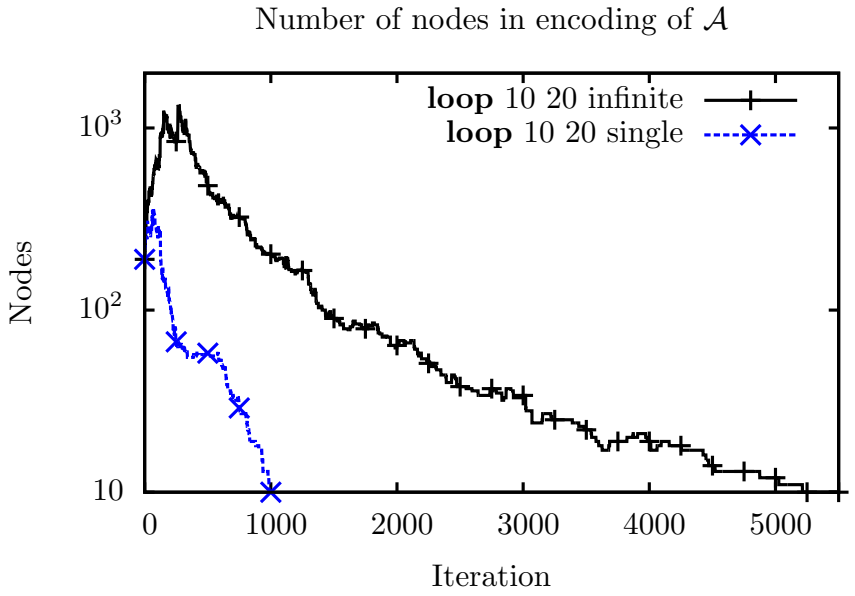


Figure 3.7: Number of nodes in decision diagram encoding of set $\mathcal{A}$ during execution of coupling phase.

Description of tested models

Model phil N Model **phil** N is a stochastic Petri net for the dining philosophers problem. In this model the starvation is avoided by the atomic taking of the two forks placed near the philosopher. The whole net comprises N philosophers. In Figure 3.8 we show the SPN associated with philosopher i, with $1 \leq i \leq N$. The initial marking has one token for each place Think_i and one token for each place Fork_i, with $1 \leq i \leq N$.

Model rphil N Analogously to model **phil** N, also **rphil** N is an SPN modeling a solution for avoiding the starvation in the problem of the dining philosophers. In

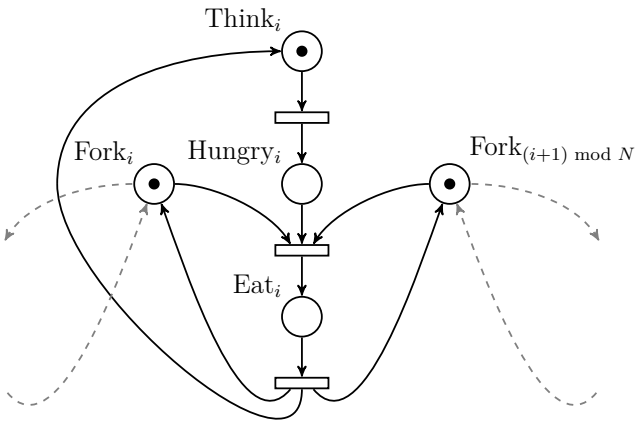


Figure 3.8: Model for the i-th philosopher in the dining philosophers net **phil** N.

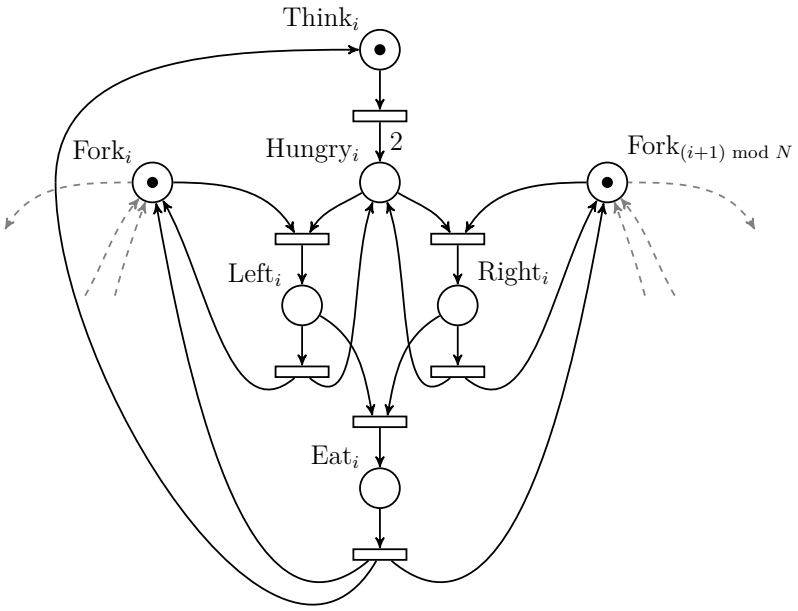


Figure 3.9: Model for the i-th philosopher in the dining philosophers net **rphil** N.

this case we allow the philosopher to take the two forks separately. However, when a philosopher is in state Left (Right) he may either take the other fork and eat or return to the state Hungry thus avoiding the starvation. The SPN consists of N models of dining philosophers as the one shown in Figure 3.9.

Model slot N Model **slot** N, shown in Figure 3.10, is an SPN with $10N$ places and $10N$ transitions, modeling a slotted ring protocol with N nodes. The model was taken from [49].

Model contention N M This model is an SPN modeling N CPUs with M processes each. All processes compete for a single global resource. One of the CPUs for this model is shown in Figure 3.11.

Model loop N M The SPN for test model **loop** consists of N nodes and N transitions forming a loop and the initial marking has M tokens in one of the places. Model **loop** 5 M is depicted in Fig. 3.12.

3.4 Application to fork-join queueing networks

In this section we test spnps on a class of models which are known to play an important role in the analysis of distributed systems and telecommunication systems: fork-join queueing networks [50, 51]. Fork-join queueing networks allow the jobs to be split into several tasks that are processed in parallel. Once all the tasks have been served, a join operation is performed and we consider the original job served.

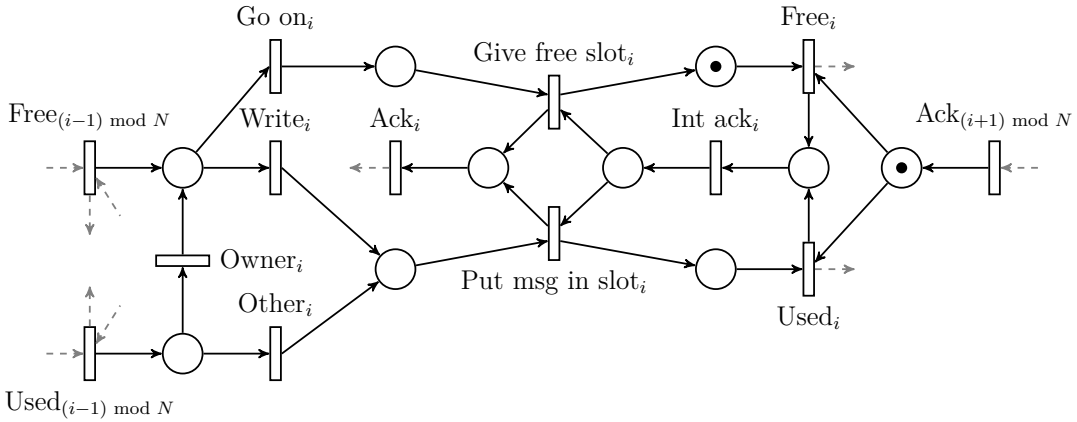


Figure 3.10: Model for the i-th node in the net **slot** N.

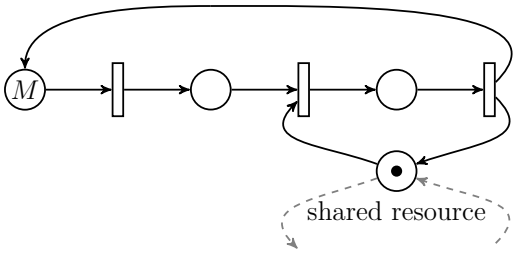


Figure 3.11: Model of a single CPU for net **contention** N M.

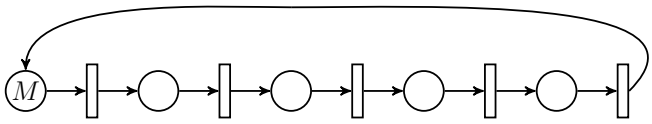


Figure 3.12: Model **loop** with $N = 5$ places and transitions and M tokens.

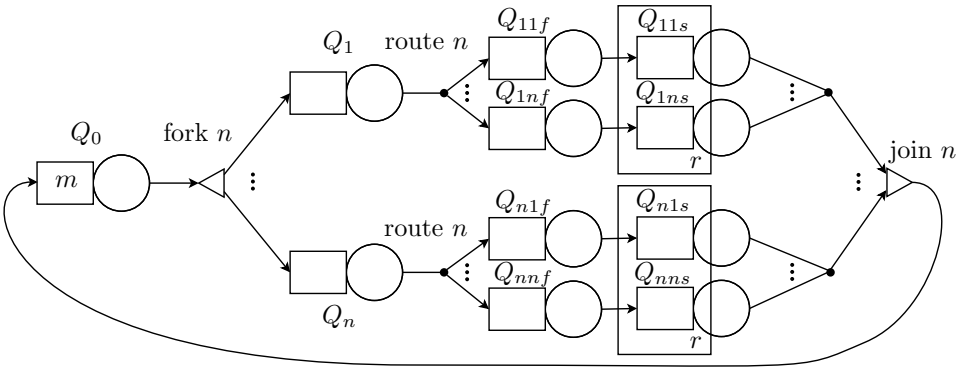


Figure 3.13: Example of a fork-join queueing network.

These models are very useful for the performance evaluation of distributed systems with task concurrency and synchronisation.

For these models very few exact results are known [52] mainly relying on product-form theory. Therefore, the definition of techniques for approximation and simulation are the principal approaches for the estimation of their stationary performance indices. In [53, 54] the authors present algorithms for perfect sampling on restricted classes of fork-join queueing networks (in the former case consisting of one node, in the latter with limitation on the probabilistic routing). They take advantage of the reachability graph properties in order to improve the efficiency of the algorithms. Similarly, another approach [18] is applicable to a different class of fork-join queueing networks; however, in the queueing networks considered there, customer routing cannot depend on the total number of customers in a set of queues, while in the models that we consider, queues can have a shared capacity (i.e., restriction on the total number of customers in a set of queues) and a customer that cannot enter a destination queue due to shared capacity being full is blocked. Further research is needed in order to assess the possibility of applying to the models considered here the general method proposed in [18]. While spnps is less efficient than these approaches for particular classes of models, its generality allows for studying SPNs (and hence fork-join queueing networks that are representable as SPNs) with a more general structure. In particular, for fork-join queueing networks, the time required for performing a perfect sampling is acceptable even for relatively large nets.

In the rest of this section we define the particular fork-join models used in testing, describe the testing procedure and discuss the performance of the tool.

3.4.1 The testing model

We consider the fork-join queueing network depicted in Figure 3.13. The model is described by parameters: n, m, r. Figure 3.13 shows the leftmost service station Q_0 that initially contains m jobs. Once a job is served, it is forked into n sub-tasks which are enqueued in $Q_1, \ldots, Q_n$, respectively. After being served by station Q_i, the sub-task i is routed to one of the stations $Q_{i1f}, \ldots, Q_{inf}$ according to a uniform probabilistic choice: this is the first phase of service. The second phase

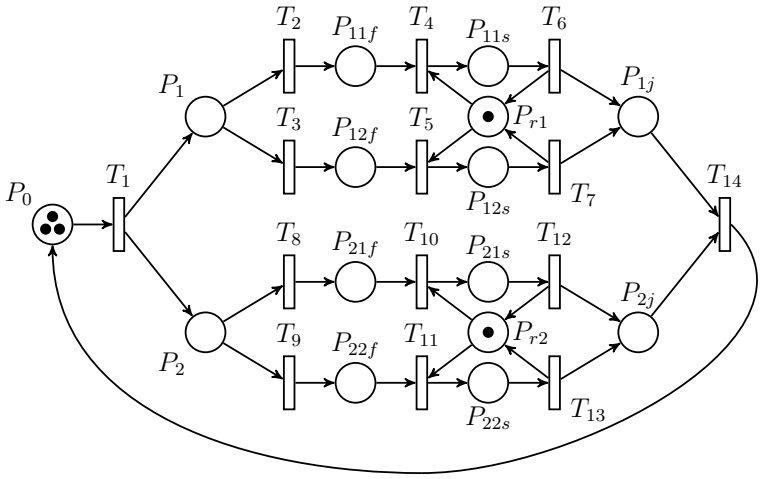


Figure 3.14: SPN associated with the fork-join queueing network of Figure 3.13 for $n = 2$, $m = 3$ and $r = 1$.

is performed by stations $Q_{i1s}, \ldots Q_{ins}$ with a constraint on the available resources: at most r sub-tasks can simultaneously be in the queues $Q_{i1s}, \ldots, Q_{ins}$ for each $i = 1, \ldots, n$. All the service times are independent and exponentially distributed and the stations are equipped with a single server. We assume a processor sharing queueing discipline.

The queueing network for parameters $n = 2$, $m = 3$, $r = 1$ is modeled by the Petri net in Figure 3.14. There are $m = 3$ tokens in place P_0, representing jobs. Transition T_1 models the fork and T_{14} the join. The resources are modelled as tokens in places P_{r1} and P_{r2}. Places P_{1j} and P_{2j} model the waiting room for sub-tasks which have been served and are waiting to be joined. The remaining places model the waiting/service rooms of the corresponding queueing stations. The resulting Petri net has $2n^2 + 3n + 1$ places and $3n^2 + 2$ transitions.

We make two important observations on the model of Figure 3.14: the first is that the SPN for $n \geq 2$ is *not* a free choice SPN. In fact, in a free choice Petri net if there is an arc from a place s to a transition t, then there must be an arc from any input place of t to any output transition of s [55]. Now, consider the net of Figure 3.14 and let s be for example P_{r1} and t be transition T_4. Consider the input place P_{11f} of T_4 and the output transition T_5 of P_{r1}: since there is no arc from P_{11f} to T_5 the net is not free choice. Therefore, the optimised algorithms for perfect sampling of free choice SPNs cannot be applied (see [54] and the references therein). The second observation is on the semantics of the join specified in terms of SPN. In fact T_{14} does not guarantee that the join occurs among the sibling sub-tasks since a sub-task may overtake another sub-task that was previously forked. The more detailed representation of the join operation requires a much more complicated stochastic process and is out of the scope of this thesis.

3.4.2 Performance evaluation

To test the performance of spnps on the fork-join model we performed two groups of tests.

In the first group of tests, we study how the performance scales with number of initial jobs. We fixed the size of the SPN by setting $n = 2$ and run the tests for parameter m ranging from 2 to 40 with step 2 and with number of resources per fork equal to half the number of jobs, $r = m/2$.

In the second group, we examine how the performance scales with the size of the SPN by fixing the number of jobs to $m = 2$ and number of resources per fork to $r = 1$ and running the tests for parameter n ranging from 2 to 16.

For each of these $20 + 15 = 35$ sets of parameters, 50 independent tests were run by performing the perfect sampling procedure with different random seeds. Rates of all transitions of the tested SPN models were set to 1. We report the results as 95% confidence intervals for the means of measured values.

Testing was done on a GNU/Linux system with Intel(R) Core(TM) i5-2310 CPU with maximum clock of 3.2 GHz and with 8GB of main memory.

The results of the testing are shown in Table 3.8 and 3.9 for the first and second batch of tests, respectively. The meaning of the columns is the following:

- N_P: number of places of the SPN.
- N_T: number of transitions of the SPN.
- $|\mathcal{RS}|$: cardinality of reachability set of the SPN.
- Init (s): time in seconds spent in generating decision diagrams that encode the reachability set and simulation update rule used in perfect sampling. If multiple samples are generated, this needs to be done only once and these decision diagrams can be reused in different sampling runs (i.e., this time can be ammortized over the sampling runs). The init times are independent of the random seed, resulting in very tight confidence intervals; we therefore report only the means.
- Sampling (s): 95% confidence interval for mean time in seconds spent in a perfect sampling run.
- Memory (kB): 95% confidence interval for mean peak memory use in kilobytes for the entire program. The program loads the SPN from a PNML file, parses it and runs the perfect sampling. The non-sampling parts use about 5MB of memory.

Common magnitudes of intervals' boundaries are shown outside the intervals.

We observe that the tool scales very well with the size $|\mathcal{RS}|$ of the reachability set when we vary the size of the net (second group of tests), but not as well when we vary the size of the initial marking (first group of tests). Figure 3.15 shows log-log plots of memory use on the top and sampling time on the bottom as functions of the reachability set size. For the model from the first group of tests with largest tested reachability set size $|\mathcal{RS}| \approx 4.857 \times 10^{12}$, the memory consumption is comparable

m	r	$\lvert\mathcal{RS}\rvert$	Init (s)	Sampling (s)	Memory (kB)
2	1	361	3.204×10^{-3}	$[3.735, 4.489] \times 10^{-3}$	$[5.638, 5.682] \times 10^{3}$
4	2	14207	5.294×10^{-3}	$[1.530, 1.754] \times 10^{-2}$	$[5.900, 5.945] \times 10^{3}$
6	3	214492	8.835×10^{-3}	$[4.449, 4.929] \times 10^{-2}$	$[6.618, 6.659] \times 10^{3}$
8	4	1.865×10^{6}	1.372×10^{-2}	$[1.068, 1.181] \times 10^{-1}$	$[7.836, 7.981] \times 10^{3}$
10	5	1.130×10^{7}	2.134×10^{-2}	$[2.250, 2.472] \times 10^{-1}$	$[7.897, 8.074] \times 10^{3}$
12	6	5.290×10^{7}	3.101×10^{-2}	$[4.277, 4.707] \times 10^{-1}$	$[1.053, 1.062] \times 10^{4}$
14	7	2.045×10^{8}	4.568×10^{-2}	$[7.911, 8.528] \times 10^{-1}$	$[1.057, 1.141] \times 10^{4}$
16	8	6.809×10^{8}	6.338×10^{-2}	$[1.434, 1.568]$	$[1.546, 1.606] \times 10^{4}$
18	9	2.012×10^{9}	8.897×10^{-2}	$[2.255, 2.441]$	$[1.583, 1.597] \times 10^{4}$
20	10	5.392×10^{9}	1.185×10^{-1}	$[3.861, 4.229]$	$[2.072, 2.379] \times 10^{4}$
22	11	1.332×10^{10}	1.554×10^{-1}	$[5.724, 6.221]$	$[2.721, 2.747] \times 10^{4}$
24	12	3.071×10^{10}	1.998×10^{-1}	$[9.056, 9.640]$	$[2.739, 3.011] \times 10^{4}$
26	13	6.673×10^{10}	2.606×10^{-1}	$[1.286, 1.389] \times 10^{1}$	$[3.620, 4.244] \times 10^{4}$
28	14	1.378×10^{11}	3.263×10^{-1}	$[1.854, 2.021] \times 10^{1}$	$[4.916, 5.118] \times 10^{4}$
30	15	2.719×10^{11}	4.086×10^{-1}	$[2.607, 2.824] \times 10^{1}$	$[5.150, 5.164] \times 10^{4}$
32	16	5.158×10^{11}	5.004×10^{-1}	$[3.521, 3.757] \times 10^{1}$	$[5.207, 5.737] \times 10^{4}$
34	17	9.446×10^{11}	6.199×10^{-1}	$[4.481, 4.871] \times 10^{1}$	$[6.514, 7.456] \times 10^{4}$
36	18	1.676×10^{12}	7.517×10^{-1}	$[6.095, 6.723] \times 10^{1}$	$[8.265, 8.833] \times 10^{4}$
38	19	2.890×10^{12}	9.036×10^{-1}	$[8.086, 8.860] \times 10^{1}$	$[9.042, 9.256] \times 10^{4}$
40	20	4.857×10^{12}	1.080	$[1.011, 1.106] \times 10^{2}$	$[9.651, 9.976] \times 10^{4}$

Table 3.8: First group of tests with $n = 2$, m ranging from 2 to 40 and $r = m/2$. The net has 15 places and 14 transitions.

n	N_P	N_T	$\lvert\mathcal{RS}\rvert$	Init (s)	Sampling (s)	Memory (kB)
2	15	14	361	3.204×10^{-3}	$[3.589, 4.216] \times 10^{-3}$	$[5.645, 5.679] \times 10^{3}$
3	28	29	27513	9.316×10^{-3}	$[2.081, 2.523] \times 10^{-2}$	$[6.654, 6.691] \times 10^{3}$
4	45	50	4.111×10^{6}	2.401×10^{-2}	$[7.792, 9.251] \times 10^{-2}$	$[8.101, 8.181] \times 10^{3}$
5	66	77	9.927×10^{8}	5.041×10^{-2}	$[2.329, 2.687] \times 10^{-1}$	$[1.123, 1.129] \times 10^{4}$
6	91	110	3.513×10^{11}	9.699×10^{-2}	$[6.775, 7.887] \times 10^{-1}$	$[1.698, 1.726] \times 10^{4}$
7	120	149	1.714×10^{14}	1.656×10^{-1}	$[1.572, 1.925]$	$[2.754, 2.771] \times 10^{4}$
8	153	194	1.103×10^{17}	2.890×10^{-1}	$[3.153, 3.755]$	$[3.229, 3.460] \times 10^{4}$
9	190	245	9.065×10^{19}	5.198×10^{-1}	$[5.843, 6.849]$	$[5.155, 5.226] \times 10^{4}$
10	231	302	9.261×10^{22}	7.359×10^{-1}	$[1.087, 1.298] \times 10^{1}$	$[9.214, 9.395] \times 10^{4}$
11	276	365	1.152×10^{26}	1.223	$[1.715, 2.031] \times 10^{1}$	$[1.009, 1.025] \times 10^{5}$
12	325	434	1.714×10^{29}	1.616	$[2.683, 3.184] \times 10^{1}$	$[1.522, 1.585] \times 10^{5}$
13	378	509	3.006×10^{32}	2.558	$[3.772, 4.537] \times 10^{1}$	$[1.888, 1.920] \times 10^{5}$
14	435	590	6.141×10^{35}	3.175	$[6.133, 7.234] \times 10^{1}$	$[2.101, 2.114] \times 10^{5}$
15	496	677	1.445×10^{39}	4.103	$[8.546, 9.790] \times 10^{1}$	$[2.653, 3.148] \times 10^{5}$
16	561	770	3.882×10^{42}	6.012	$[1.067, 1.259] \times 10^{2}$	$[3.798, 4.173] \times 10^{5}$

Table 3.9: Second group of tests with n ranging from 2 to 16, $m = 2$ and $r = 1$.

to the model from the second group of tests for which $|\mathcal{RS}| \approx 1.152 \times 10^{26}$, and the sampling time is comparable to the model from the second group of tests with the largest tested $|\mathcal{RS}| \approx 3.882 \times 10^{42}$.

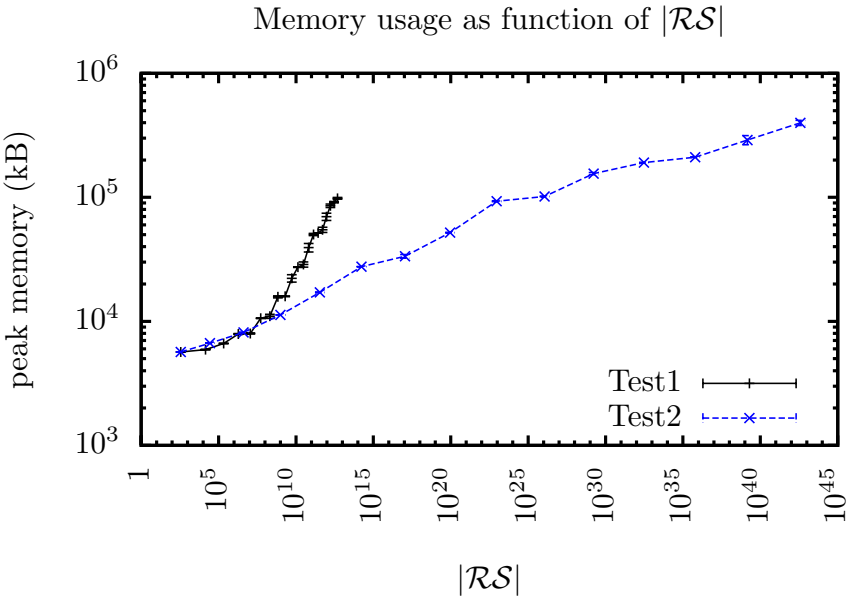
As performance of the algorithm depends in large part on efficiency of MDDs that are used to encode subsets of the reachability set and the simulation update rule, and efficiency of MDDs depends on the properties of the sets or functions that they encode, the observed difference in performance between the two groups of tests is likely in part due to the structure of the reachability sets. If we consider the reachability sets of the n SPN components between fork and join transitions, then for $m = 1$ the reachability set of the entire SPN is equal to the union of a singleton set containing initial marking (where the single token is in the initial place) and the Cartesian product of the n components' reachability sets restricted to the case when the initial place is empty. This is highly structured and is well suited to representation using decision diagrams (but note that the high symmetry of the reachability sets is not exploited in the decision diagram representation, only the high decomposability; i.e., the important property is that the reachability set is similar to a Cartesian product and not that the sets in the Cartesian product are equal). Similar considerations apply for other small values of m. In contrast, for small n and large m, the reachability set is not as well structured (we obtain a large union of $m + 1$ Cartesian products over components).

In addition, because the implementation encodes reachability sets and their subsets by decision diagrams in which levels correspond to Petri net places, enlarging the net makes the decision diagrams taller (more levels), while enlarging the marking makes them wider (more arcs per node). Since the decision diagrams gain efficiency, in comparison to a tree representation of the reachability set, by reusing nodes on lower levels, this makes them likely to be more efficient in the former case—in taller diagrams there are more levels and reusing nodes deeper in the diagram is often more profitable for efficiency.

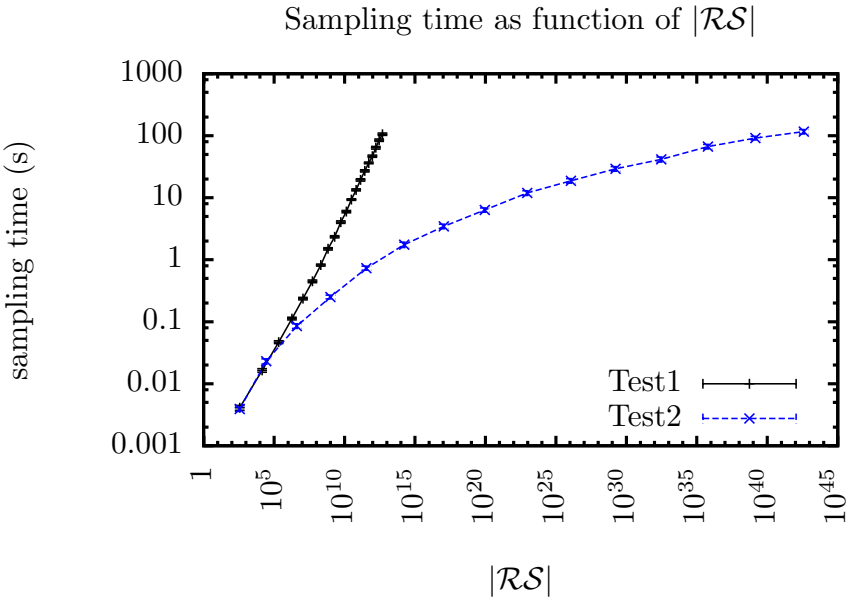
3.5 Summary

In this chapter we have introduced a perfect sampling algorithm for SPNs that, in contrast to previous work, does not require any restrictions on the structure of the analysed nets. The only required restrictions are the finiteness of the reachability set and a restriction on the type of marking-dependence of transition firing rates (however, common semantics such as single server and infinite server can be accommodated). The algorithm achieves its generality by encoding the reachability set and transition functions of an SPN using MDDs, implementing coupling from the past efficiently without relying on special structure of the SPN.

Next, we have presented a tool, called spnps, that implements the described algorithm and tested its performance on several models, ending the chapter with a test on fork-join queueing networks. While the time and space complexities of the algorithm are in the worst case more than exponential in the size of the SPN, the tests show that the performance is acceptable in many cases even for very large state spaces.



(a) Memory consumption



(b) Time consumption

Figure 3.15: Comparison of the memory usage and the sampling time for the first and the second batch of tests.

4

Computation of normalising constant for product-form stochastic Petri nets

A special class of product-form models has been identified first in QNs [56, 57] and later in SPNs [10, 11, 12, 13] and other modeling formalisms; for a model in this class, its stationary probability distribution can be obtained in the form of a product over model components. State probabilities for these models are usually obtained in an unnormalised form (formally, a measure over the state space of the stochastic process is obtained), requiring computation of the normalising constant which in principle can be directly computed as a sum of unnormalised probabilities over the state space of the underlying CTMC or using more efficient convolution algorithms [58, 22]. Because of the simple form of the stationary probability distribution, the stationary analysis of product-form models is greatly simplified in comparison to other models and more complex models with larger state spaces can be efficiently analysed using special algorithms such as mean value analysis (MVA) algorithm [23] for performance evaluation of product-form SPNs. However, convolution algorithm and MVA for SPNs require a special structure of the reachability set: they can only be applied to the special class of S-invariant reachable SPNs. Furthermore, at the state of the art it is not known how to automatically check the membership of this class without generating the reachability set. In addition, the convolution algorithm for SPNs is less efficient than in the case of QNs due to the need to solve a large number of systems of linear Diophantine equations.

In this chapter we propose a formalism-agnostic recursive algorithm for computation of the normalising constant for general product-form models with finite state spaces. In contrast to the convolution algorithm for SPNs, we need to generate the reachability set of the analysed SPN, but we do not require S-invariant reachability and the proposed algorithm can thus be applied to general SPNs with finite reachability sets. Also, we do not need to solve systems of Diophantine equations and performance of the proposed algorithm is thus much better than the convolution algorithm if reachability set generation is not taken into account, while being comparable otherwise. We also propose related methods for computation of performance measures of product-form SPNs with finite reachability sets.

Section 4.1 introduces the algorithm for computation of normalising constant for product-form models. In Section 4.2 we compare the proposed algorithm to the

convolution algorithm for a class of S-invariant reachable product-form SPNs [22] and in Section 4.3 we propose methods for performance evaluation based on the proposed algorithm, comparing them to MVA. Finally, in Section 4.4 we report results of computational experiments on SPNs in which we test performance of the proposed algorithm and compare it to the performance of the convolution algorithm, showing that the proposed algorithm compares favourably to convolution.

4.1 Computation of normalising constant – algorithm MDD-rec

We assume throughout this chapter that the model under consideration is decomposed into K submodels, $K \in \mathbb{N}$. State space of the model is assumed to be finite, and (also finite) local state spaces of the submodels are denoted by $\mathcal{S}_K, \ldots, \mathcal{S}_1$. Local state space $\mathcal{S}_k$ of cardinality $n_k \in \mathbb{N}$ is identified with the set $\{0, 1, \ldots, n_k - 1\}$. Cartesian product $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$ of local state spaces is denoted by $\hat{\mathcal{S}}$ and called *potential state space* and state space of the model is denoted by $\mathcal{S}$; in general $\mathcal{S} \subseteq \hat{\mathcal{S}}$. State of the model is denoted by a K-tuple of integers $(s_K, \ldots, s_1) \in \mathcal{S}$, where $s_k \in \mathcal{S}_k$ is the local state of the submodel k.

As explained in Chapter 2, a state space $\mathcal{S}$ of a model that is decomposed into K submodels as above, can be encoded using a multi-valued decision diagram (MDD).

In this chapter, models with product-form stationary probability distribution are taken into consideration. Such models are defined by their state probabilities being products over submodels of the probabilities of submodels' states, i.e. stationary probability $\boldsymbol{\pi}_{\boldsymbol{s}}$ of an arbitrary state $\boldsymbol{s} = (s_K, \ldots, s_1) \in \mathcal{S}$ can be computed as

$$\boldsymbol{\pi}_{\boldsymbol{s}} = \frac{1}{G} \prod_{k=K}^{1} g_k(s_k) \tag{4.1}$$

where G is a normalising constant needed for the probabilities to sum to 1:

$$G = \sum_{\boldsymbol{s} \in \mathcal{S}} \prod_{k=K}^{1} g_k(s_k) \tag{4.2}$$

and $g_k : \mathcal{S}_k \to \mathbb{R}_{\geq 0}, k = K, \ldots, 1$ are functions defining (not necessarily normalised) probabilities of local states of the submodels. One of the main problems in the analysis of product-form models is efficient computation of the normalising constant G. A recursive algorithm for its computation is described in this section. This algorithm computes the normalising constant by walking over nodes of an MDD that encodes the state space $\mathcal{S}$ of the model. While in the theoretical exposition of the algorithm in this chapter we assume that the MDD encoding $\mathcal{S}$ is taken as input to the algorithm, in the implementation we efficiently generate this MDD by using the same encoding and saturation algorithm [42] described in Section 2.2.

Definition 4.1.1. Mass of an MDD node *Let $M : MDD \to \mathbb{R}_{\geq 0}$ be a function on the nodes of the MDD encoding $\mathcal{S}$, defined as follows. For an arbitrary MDD node*

$\langle l.p\rangle$, *its* mass $M(\langle l.p\rangle)$ *is defined as*

$$M(\langle l.p\rangle) = \begin{cases} \sum\limits_{\boldsymbol{s}\in\mathcal{B}(\langle l.p\rangle)} \prod\limits_{k=l}^{1} g_k(s_k) & \text{if } l > 0, \\ p & \text{otherwise.} \end{cases} \tag{4.3}$$

From the definitions of M, $\mathcal{B}$ and G it can be easily seen that the mass of the root node $\langle K.r\rangle$ of the MDD is equal to the normalising constant G:

$$M(\langle K.r\rangle) = \sum_{\boldsymbol{s}\in\mathcal{B}(\langle K.r\rangle)} \prod_{k=K}^{1} g_k(s_k) = \sum_{\boldsymbol{s}\in\mathcal{S}} \prod_{k=K}^{1} g_k(s_k) = G. \tag{4.4}$$

Thus, computing G can be performed by computing the mass of the root node of the MDD. The rest of this section is concerned with efficient computation of the mass $M(\langle K.r\rangle)$.

In general, mass $M(\langle l.p\rangle)$ of an arbitrary non-terminal node can be obtained as a weighted sum of masses of its neighbouring nodes in the level $l-1$:

$$M(\langle l.p\rangle) = \sum_{\boldsymbol{s}\in\mathcal{B}(\langle l.p\rangle)} \prod_{k=l}^{1} g_k(s_k) = \sum_{\substack{s_l\in\mathcal{S}_l \\ \boldsymbol{t}\in\mathcal{B}(\langle l.p\rangle[s_l])}} g_l(s_l) \prod_{k=l-1}^{1} g_k(t_k) =$$

$$= \sum_{s_l\in\mathcal{S}_l} g_l(s_l) \sum_{\boldsymbol{t}\in\mathcal{B}(\langle l.p\rangle[s_l])} \prod_{k=l-1}^{1} g_k(t_k) = \sum_{s_l\in\mathcal{S}_l} g_l(s_l) M(\langle l.p\rangle[s_l]).$$

This recurrence relation leads to a recursive algorithm for computation of the mass $M(\langle K.r\rangle)$. The algorithm recursively walks over the nodes of the MDD encoding the set $\mathcal{S}$, computing masses of MDD nodes and caching them in node labels, as depicted in Algorithm 6. The computation of normalising constant G is then simply a matter of invoking MDD-rec($\langle K.r\rangle$, MDD, NONE, $g_K, \ldots, g_1$) on the root note $\langle K.r\rangle$ of an MDD encoding state space $\mathcal{S}$ of the model, with labeling function L set to sentinel value NONE for all nodes, $L \equiv$ NONE.

Generating the MDD that encodes the reachability set of an SPN is in EXPSPACE—in the worst case the size of the resulting MDD is exponential in the size of the SPN. Because MDD-rec visits all nodes of this MDD, it is also in EXPSPACE.

4.2 Comparison of MDD-rec with convolution algorithm

In this section we first recall the convolution algorithm [22] for the computation of normalising constant of the product-form SPNs from the class of S-invariant reachable SPNs and compare it with the proposed algorithm MDD-rec. We show that, in terms of data flow, the two algorithms are equivalent on the class of S-invariant reachable product-form SPNs with finite state spaces. While the convolution algorithm can handle SPNs with unbounded state spaces (but only under further assumption that closed form solutions can be obtained for the infinite sums that appear in the computation of the normalising constant), MDD-rec can handle a larger class of finite state space SPNs.

Algorithm 6: MDD-rec($\langle l.p\rangle$, MDD, $L, g_K, \ldots, g_1$)

Data:
node $\langle l.p\rangle$ of the MDD
multi-valued decision diagram MDD encoding set $\mathcal{S}$
node labels $L : \text{MDD} \to \mathbb{R}_{\geq 0} \cup \{\text{NONE}\}$
functions $g_k : \mathcal{S}_k \to \mathbb{R}_{\geq 0}, k = K, \ldots, 1$ defining the product-form
Result: mass $M(\langle l.p\rangle)$, modified node labels L

```
1 begin
2     if l = 0 then
3         return p
4     if L(⟨l.p⟩) = NONE then
5         L(⟨l.p⟩) ← Σ_{s_l ∈ S_l} g_l(s_l) MDD-rec(⟨l.p⟩[s_l], MDD, L, g_K, ..., g_1)
6     return L(⟨l.p⟩)
```

4.2.1 Convolution algorithm for computation of normalising constant for stochastic Petri nets

The convolution algorithm [22] for computation of the normalising constant assumes a particular structure of the reachability set that allows for a recursive decomposition. Let $\boldsymbol{S}$ be the matrix with rows comprised in all minimal support S-invariants of the SPN, and let $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ be *load vector* of the SPN. SPN is *S-invariant reachable* if its reachability set $\mathcal{RS}$ satisfies the following condition:

$$\forall \boldsymbol{m} \in \mathbb{N}^n,\ \boldsymbol{m} \in \mathcal{RS} \iff \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}. \tag{4.5}$$

For any subset of SPN places $\mathcal{P}' \subseteq \mathcal{P}$ and any vector $\boldsymbol{W} \in \mathbb{R}^n$, set $\mathcal{E}(\mathcal{P}', \boldsymbol{W}) \subseteq \mathbb{N}^n$ of markings is defined with

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \{\boldsymbol{m} \in \mathbb{N}^n : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} \text{ and } \forall p \in \mathcal{P} \setminus \mathcal{P}', m_p = 0\}. \tag{4.6}$$

It can be easily seen that $\mathcal{RS} = \mathcal{E}(\mathcal{P}, \boldsymbol{V})$. Given a set of markings $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$, marking set $M_p(\mathcal{P}', \boldsymbol{W})$ of place $p \in \mathcal{P}$ is defined as

$$M_p(\mathcal{P}', \boldsymbol{W}) = \{i \in \mathbb{N} : \exists \boldsymbol{m} \in \mathcal{E}(\mathcal{P}', \boldsymbol{W}) \text{ such that } m_p = i\}. \tag{4.7}$$

This is the set of all different markings for place p that appear in the given set of markings.

The following Lemma appears in [59] but in a different formulation, so we include the proof here for clarity.

Lemma 4.2.1. *For any S-invariant reachable SPN $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$ with load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$, the following decomposition of set $\mathcal{E}(\mathcal{P}', \boldsymbol{W})$ holds for all subsets $\mathcal{P}' \subseteq \mathcal{P}$, arbitrary place $p \in \mathcal{P}'$ and all vectors $\boldsymbol{W}$, $\boldsymbol{0} \leq \boldsymbol{W} \leq \boldsymbol{V}$:*

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}' \setminus \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \right], \tag{4.8}$$

where the union is over mutually disjoint sets (i.e., the sets that appear in the union on the right-hand side define a partition of the set that appears on the left-hand side). Here $\boldsymbol{e}_p$ is a vector of length n with all elements equal to zero except the p-th element which is equal to one, and $\boldsymbol{S}_p$ is the p-th column of the matrix $\boldsymbol{S}$.

Proof.

$$\mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \{\boldsymbol{m} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} \text{ and } \forall q \in \mathcal{P} \setminus \mathcal{P}', m_q = 0\} =$$

$$= \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \{\boldsymbol{m} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} \text{ and } m_p = i \text{ and } \forall q \in \mathcal{P} \setminus \mathcal{P}', m_q = 0\} =$$

$$= \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \{i\boldsymbol{e}_p + \boldsymbol{m} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{W} - i\boldsymbol{S}_p \text{ and } m_p = 0 \text{ and } \forall q \in \mathcal{P} \setminus \mathcal{P}', m_q = 0\} =$$

$$= \bigcup_{i \in M_p(\mathcal{P}', \boldsymbol{W})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P}' \setminus \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p)\right].$$

Clearly this union is over mutually disjoint sets. □

In particular, note that the reachability set $\mathcal{RS}$ can be partitioned by conditioning on the number of tokens in some place $p \in \mathcal{P}$:

$$\mathcal{RS} = \mathcal{E}(\mathcal{P}, \boldsymbol{V}) = \bigcup_{i \in M_p(\mathcal{P}, \boldsymbol{V})} \left[i\boldsymbol{e}_p + \mathcal{E}(\mathcal{P} \setminus \{p\}, \boldsymbol{V} - i\boldsymbol{S}_p)\right], \tag{4.9}$$

and all of the sets $\mathcal{E}(\mathcal{P} \setminus \{p\}, \boldsymbol{V} - i\boldsymbol{S}_p), i \in M_p(\mathcal{P}, \boldsymbol{V})$ that appear on the right-hand side can be further partitioned in the same manner. This yields a recursive state space decomposition scheme that is the basis for the convolution algorithm for computation of the normalising constant. Note that this decomposition exists only for the S-invariant reachable SPNs.

For product-form S-invariant reachable nets, a recurrence relation, derived from the decomposition of the state space, is used as the basis of convolution algorithm. For any nonempty subset $\mathcal{P}' \subseteq \mathcal{P}$ of the set of places, and any vector $\boldsymbol{W}$, $\boldsymbol{0} \leq \boldsymbol{W} \leq \boldsymbol{V}$, the following value is defined:

$$G(\mathcal{P}', \boldsymbol{W}) = \sum_{\boldsymbol{m} \in \mathcal{E}(\mathcal{P}', \boldsymbol{W})} \prod_{p \in \mathcal{P}'} g_p(m_p). \tag{4.10}$$

Obviously, for an S-invariant reachable product-form net with set of places $\mathcal{P}$ and load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$, $G(\mathcal{P}, \boldsymbol{V})$ is equal to the normalising constant G. From the lemma it directly follows that for a nonempty subset $\mathcal{P}' \subseteq \mathcal{P}$ of the set of places, any place $p \in \mathcal{P}'$ and any vector $\boldsymbol{W}$, $\boldsymbol{0} \leq \boldsymbol{W} \leq \boldsymbol{V}$, the following recurrence relation holds:

$$G(\mathcal{P}', \boldsymbol{W}) = \sum_{i \in M_p(\mathcal{P}', \boldsymbol{W})} g_p(i) G(\mathcal{P}' \setminus \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p), \tag{4.11}$$

and the boundary conditions can be derived from the definition of $G(\cdot, \cdot)$:

$$\begin{array}{c} G(\{p\}, \boldsymbol{W}) = \sum_{i \in M_p(\{p\}, \boldsymbol{W})} g_p(i), \forall p \in \mathcal{P}, \forall \boldsymbol{W} \text{ such that } \boldsymbol{0} \leq \boldsymbol{W} \leq \boldsymbol{V}; \\ G(\mathcal{P}', \boldsymbol{W}) = 0, \forall \mathcal{P}' \subseteq \mathcal{P}, \forall \boldsymbol{W} \text{ such that } \mathcal{E}(\mathcal{P}', \boldsymbol{W}) = \emptyset. \end{array} \tag{4.12}$$

The convolution algorithm is a recursive algorithm that is based on the above recurrence relation; assuming some ordering of the set of places $\mathcal{P}$, it computes the

normalising constant $G = G(\mathcal{P}, \boldsymbol{V})$ by conditioning on the number of tokens in the first place p, as in recurrence relation (4.11). Values $G(\mathcal{P} \setminus \{p\}, \boldsymbol{V} - i\boldsymbol{S}_p)$ needed in the computation are computed by recursively applying the same recurrence relation, until one of the boundary conditions (4.12) is reached. The convolution algorithm for SPNs has two drawbacks:

- It can only be applied to S-invariant reachable SPNs, and verification of S-invariant reachability in general requires either generation of the reachability set or a manual proof.
- It computes marking sets $M_p(\mathcal{P}', \boldsymbol{W})$ for all combinations of places p, subsets $\mathcal{P}'$ of the set of places and vectors $\boldsymbol{W}$ that are encountered during the recursive computation. This in general requires solving systems of linear Diophantine equations—where solutions are sought only in the set of integers—which is a difficult problem and in principle represents a serious limitation of the algorithm. For practical models, the systems in question are often feasible and the marking sets can sometimes be manually deduced from the definition of the model.

Later in this section, we present in Proposition 4.2.2 a method for computing the marking sets by determining feasibility of certain integer linear programming problems (ILP); this is also a difficult problem (in fact, it is NP-complete [60]) but as shown in Section 4.4 for tested models the obtained ILPs are manageable.

4.2.2 Comparison of MDD-rec with the convolution algorithm for stochastic Petri nets

In the following, MDDs with levels corresponding to SPN places will be used to encode the reachability set of the SPN. Since arc labels with source nodes in levels $l \in \{n, \ldots, 1\}$ of the MDD are confined to sets $\{0, 1, \ldots, k_l\}$ for some numbers $k_l \in \mathbb{N}$, functions $\phi_l : M_l(\mathcal{P}, \boldsymbol{V}) \to \{0, \ldots, |M_l(\mathcal{P}, \boldsymbol{V})| - 1\}, l \in \{n, \ldots, 1\}$ are defined that rename SPN place markings into MDD arc labels:

$$\phi_l(i) = |\{j \in M_l(\mathcal{P}, \boldsymbol{V}) : j < i\}|, \forall i \in M_l(\mathcal{P}, \boldsymbol{V}). \tag{4.13}$$

Function ϕ_l can be seen to map marking i of place l to its ordinal number in the set $M_l(\mathcal{P}, \boldsymbol{V})$ of all possible markings of place l. Since by assumption all places are bounded, these functions are well defined. For $l \in \{1, \ldots, n\}$, where n is the number of SPN places as before, sets of places $\mathcal{P}_l \subseteq \mathcal{P}$ are defined with $\mathcal{P}_l = \{P_1, \ldots, P_l\}$, and sets of renamed submarkings $\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W}) \subseteq \bigtimes_{i=l}^{1} \{0, \ldots, |M_i(\mathcal{P}, \boldsymbol{V})| - 1\}$ for a vector $\boldsymbol{W} \in \mathbb{R}^n$ are defined with

$$\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W}) = \{(\phi_l(m_l), \ldots, \phi_1(m_1)) : \boldsymbol{m} \in \mathcal{E}(\mathcal{P}_l, \boldsymbol{W})\}. \tag{4.14}$$

With these definitions, from Lemma 4.2.1 the following formulation of the renamed state space decomposition directly follows. For all levels $l \in \{1, \ldots, n\}$ and all vectors $\boldsymbol{W}$ such that $\boldsymbol{0} \leq \boldsymbol{W} \leq \boldsymbol{V}$,

$$\mathcal{S}_\phi(\mathcal{P}_l, \boldsymbol{W}) = \bigcup_{i \in M_l(\mathcal{P}_l, \boldsymbol{W})} (\phi_l(i)) \mathcal{S}_\phi(\mathcal{P}_{l-1}, \boldsymbol{W} - i\boldsymbol{S}_l). \tag{4.15}$$

Lemma 4.2.2. *Let an ordered quasi-reduced MDD over the renamed potential reachability set* $\hat{\mathcal{S}} = \bigtimes_{i=n}^{1}\{0,\ldots,|M_i(\mathcal{P},\boldsymbol{V})|-1\}$ *with* n *levels corresponding to Petri net places encode the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n,\boldsymbol{V}) = \{(\phi_n(m_n),\ldots,\phi_1(m_1)) \in \hat{\mathcal{S}} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}\}$ *of an S-invariant reachable stochastic Petri net with* n *places, load vector* $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ *and matrix of minimal support S-invariants* $\boldsymbol{S}$*. Assume that a non-terminal node* $\langle l.\boldsymbol{W}\rangle$ *on level greater than* 1*,* $l > 1$*, encodes set* $\mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W})$ *for a subset of places* $\mathcal{P}_l \subseteq \mathcal{P}$ *and some vector* $\boldsymbol{W}$*,* $\boldsymbol{0} \le \boldsymbol{W} \le \boldsymbol{V}$*, that is* $\mathcal{B}(\langle l.\boldsymbol{W}\rangle) = \mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W})$*.*

Then there exist nodes on level $l-1$ *that encode sets from family* $\{\mathcal{S}_\phi(\mathcal{P}_{l-1}, \boldsymbol{W} - i\boldsymbol{S}_l) : i \in M_l(\mathcal{P}_l,\boldsymbol{W})\}$*; denote these nodes with* $\langle l-1.\boldsymbol{W} - i\boldsymbol{S}_l\rangle, i \in M_l(\mathcal{P}_l,\boldsymbol{W})$*, respectively. Destination nodes of outgoing arcs of node* $\langle l.\boldsymbol{W}\rangle$ *are:*

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle l-1.\boldsymbol{W} - i\boldsymbol{S}_l\rangle & \text{if } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise.} \end{cases} \tag{4.16}$$

Proof. From the following equalities

$$\begin{array}{l} \mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W}) = \bigcup_{i\in M_l(\mathcal{P}_l,\boldsymbol{W})}(\phi_l(i))\mathcal{S}_\phi(\mathcal{P}_{l-1},\boldsymbol{W} - i\boldsymbol{S}_l) \\ \quad\quad\| \\ \mathcal{B}(\langle l.\boldsymbol{W}\rangle) = \bigcup_{i\in M_l(\mathcal{P},\boldsymbol{V})}(\phi_l(i))\mathcal{B}(\langle l.\boldsymbol{W}\rangle[\phi_l(i)]) \end{array} \tag{4.17}$$

and because ϕ_l is an injective function, it easily follows that

$$\mathcal{B}(\langle l.\boldsymbol{W}\rangle[\phi_l(i)]) = \begin{cases} \mathcal{S}_\phi(\mathcal{P}_{l-1},\boldsymbol{W} - i\boldsymbol{S}_l) & \text{if } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \emptyset & \text{otherwise.} \end{cases} \tag{4.18}$$

Therefore there exist nodes on level $l-1$ that encode sets from family $\{\mathcal{S}_\phi(\mathcal{P}_{l-1}, \boldsymbol{W} - i\boldsymbol{S}_l) : i \in M_l(\mathcal{P}_l,\boldsymbol{W})\}$; denote these nodes as in the statement of the lemma and the configuration of the arcs follows. □

Proposition 4.2.1. Structure of an MDD encoding state space of an S-invariant reachable SPN *Let an ordered quasi-reduced MDD over the renamed potential reachability set* $\hat{\mathcal{S}} = \bigtimes_{i=n}^{1}\{0,\ldots,|M_i(\mathcal{P},\boldsymbol{V})|-1\}$ *with* n *levels corresponding to Petri net places encode the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n,\boldsymbol{V}) = \{(\phi_n(m_n),\ldots,\phi_1(m_1)) \in \hat{\mathcal{S}} : \boldsymbol{S}\boldsymbol{m} = \boldsymbol{V}\}$ *of an S-invariant reachable stochastic Petri net with* n *places, load vector* $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ *and matrix of minimal support S-invariants* $\boldsymbol{S}$*.*

Then the structure of the MDD is as follows:

1. *Top level* n *contains a single root node, denoted here with* $\langle n.\boldsymbol{V}\rangle$*, which encodes the renamed reachability set* $\mathcal{S}_\phi(\mathcal{P}_n,\boldsymbol{V})$*.*

2. *Levels* $l \in \{n-1,\ldots,1\}$ *each contain exactly the nodes which encode sets from the family* $\{\mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{U} - i\boldsymbol{S}_{l+1}) : \exists$ *node on level* $l+1$ *that encodes* $\mathcal{S}_\phi(\mathcal{P}_{l+1},\boldsymbol{U})$ *and* $i \in M_{l+1}(\mathcal{P}_{l+1},\boldsymbol{U})\}$*. A node that encodes set* $\mathcal{S}_\phi(\mathcal{P}_l,\boldsymbol{W})$ *from this family is denoted with* $\langle l.\boldsymbol{W}\rangle$*.*

3. *Each non-terminal node* $\langle l.\boldsymbol{W}\rangle$ *has exactly* $|M_l(\mathcal{P},\boldsymbol{V})|$ *outgoing arcs with destination nodes as follows. For all* $i \in M_l(\mathcal{P},\boldsymbol{V})$*,*

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle l-1.\boldsymbol{W} - i\boldsymbol{S}_l\rangle & \text{if } l > 1 \text{ and } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \langle 0.1\rangle & \text{if } l = 1 \text{ and } i \in M_l(\mathcal{P}_l,\boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise.} \end{cases} \tag{4.19}$$

Proof. Since, by definition, root node $\langle n.\boldsymbol{V}\rangle$ is the only node at level n and it encodes set $\mathcal{S}_\phi(\mathcal{P}_n, \boldsymbol{V})$ by assumption, statement 1. holds.

We now prove statements 2. and 3. for the lower levels by finite induction, starting from the level $n-1$. For $n = 1$, statement 2. is trivial and statement 3. can be easily checked by inspection of the decision diagram. In the following we assume that $n > 1$.

Induction base By applying Lemma 4.2.2 on node $\langle n.\boldsymbol{V}\rangle$, it follows that statement 2. holds for level $n-1$ and statement 3. holds for the single node in level n.

Induction step Let $l \in \{n-1, \ldots, 2\}$ and assume that statement 2. holds for level l and that statement 3 holds for all nodes in level $l+1$. Applying the Lemma 4.2.2 to each of the nodes $\langle l.\boldsymbol{W}\rangle$ from statement 2. in turn, and noting that, due to canonicity, same sets cannot be encoded by different nodes, we obtain that statement 2. holds for level $l-1$, and that statement 3. holds for level l.

By induction, statement 2. holds for all levels $1, \ldots, n-1$ and statement 3. holds for all nodes in all levels $2, \ldots, n$.

We still need to show that statement 3. holds for all nodes in level $l = 1$. In this case, since $\mathcal{B}(\langle 1.\boldsymbol{W}\rangle) = \mathcal{S}_\phi(\mathcal{P}_1, \boldsymbol{W}) = \{\phi_1(m_1) : m_1 \in M_1(\mathcal{P}_1, \boldsymbol{W})\}$, it is easy to establish:

$$\langle l.\boldsymbol{W}\rangle[\phi_l(i)] = \begin{cases} \langle 0.1\rangle & \text{if } i \in M_l(\mathcal{P}_l, \boldsymbol{W}), \\ \langle 0.0\rangle & \text{otherwise} \end{cases} \tag{4.20}$$

which concludes the proof. □

Since the described MDD encodes the renamed reachability set of the SPN, and computation of the normalising constant requires the original place markings, we define functions $h_1, \ldots, h_n$ with:

$$h_l : \{0, \ldots, |M_l(\mathcal{P}, \boldsymbol{V})| - 1\} \to \mathbb{R},\ h_l(i) = g_l(\phi_l^{-1}(i)), \tag{4.21}$$

where $g_1, \ldots, g_n$ are the functions that define product-form (4.1). Mass of the root node is then computed using functions h_l instead of g_l to compute the normalising constant. With these definitions and from the definition of the mass of MDD node from Section 4.1, it can be easily seen that the following recurrence relation holds for an MDD that encodes the state space of an S-invariant reachable SPN:

$$M(\langle l.\boldsymbol{W}\rangle) = \sum_{i \in M_l(\mathcal{P}_l, \boldsymbol{W})} g_l(i) M(\langle l-1.\boldsymbol{W} - i\boldsymbol{S}_l\rangle). \tag{4.22}$$

After identification $M(\langle l.\boldsymbol{W}\rangle) \equiv G(\mathcal{P}_l, \boldsymbol{W})$, it is obvious that the above recurrence relation is equivalent to the recurrence relation (4.11). Further, from the described structure of the MDD, the following boundary conditions can be derived:

$$\begin{aligned} &M(\langle 1.\boldsymbol{W}\rangle) = \textstyle\sum_{i \in M_1(\mathcal{P}_1, \boldsymbol{W})} g_1(i) = G(\mathcal{P}_1, \boldsymbol{W}), \forall \boldsymbol{W} \text{ such that } \mathbf{0} \le \boldsymbol{W} \le \boldsymbol{V}; \\ &M(\langle l.\boldsymbol{W}\rangle) = 0 = G(\mathcal{P}_l, \boldsymbol{W}), \forall l \in \{1, \ldots, n\}, \forall \boldsymbol{W} \text{ such that } \mathcal{E}(\mathcal{P}_l, \boldsymbol{W}) = \emptyset. \end{aligned} \tag{4.23}$$

While these boundary conditions include only a subset of the boundary conditions (4.12), they do include all boundary conditions that can be encountered in a

convolution algorithm when computing the normalising constant of an SPN with load vector $\boldsymbol{V}$, assuming the places in the convolution algorithm are taken in the order $P_n, P_{n-1}, \ldots, P_1$. Since $M(\langle n.\boldsymbol{V}\rangle) = G = G(\mathcal{P}, \boldsymbol{V})$, from the above equivalence it follows that MDD-rec computes the normalising constant for an S-invariant reachable product-form SPN with load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$ and is equivalent to the convolution algorithm in terms of data flow. The following paragraphs discuss the differences between MDD-rec and convolution.

Computation of marking sets An MDD encoding reachability set of an S-invariant reachable SPN effectively also encodes marking sets of places that are needed in the convolution algorithm. For example, if the node $\langle l.\boldsymbol{W}\rangle$ appears in the MDD, marking set $M_l(\mathcal{P}_l, \boldsymbol{W})$ is encoded by the outgoing arcs of the node $\langle l.\boldsymbol{W}\rangle$:

$$i \in M_l(\mathcal{P}_l, \boldsymbol{W}) \iff \langle l.\boldsymbol{W}\rangle[\phi(i)] \neq \langle 0.0\rangle. \tag{4.24}$$

Thus the difficult problem of computing marking sets is here solved by generation of the reachability set. In contrast, for convolution algorithm a marking set $M_l(\mathcal{P}_l, \boldsymbol{W})$ can be obtained by considering feasibility of certain ILPs, as follows.

Proposition 4.2.2. Computing marking sets *Let $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$ be an S-invariant reachable SPN with n places and load vector $\boldsymbol{V} = \boldsymbol{S}\boldsymbol{m}_0$. Consider a nonempty subset $\mathcal{P}'$ of places, $\emptyset \neq \mathcal{P}' \subseteq \mathcal{P}$, place $p \in \mathcal{P}'$ and vector $\boldsymbol{W} \in \mathbb{N}^n$ such that $\boldsymbol{0} \leq \boldsymbol{W} \leq \boldsymbol{V}$.*

Then for every $i \in \mathbb{N}$, $i \in M_p(\mathcal{P}', \boldsymbol{W})$ if and only if ILP

$$\begin{array}{ll} \textit{maximize} & 0 \\ \textit{subject to} & \boldsymbol{S}\boldsymbol{x} = \boldsymbol{W} - i\boldsymbol{S}_p \\ & x_q = 0, \forall q \in \mathcal{P} \setminus (\mathcal{P}' \setminus \{p\}) \\ & x_q \geq 0, \forall q \in \mathcal{P}' \setminus \{p\} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{array} \tag{4.25}$$

is feasible.

Proof.

$$i \in M_p(\mathcal{P}', \boldsymbol{W}) \iff$$

$$\iff \exists \boldsymbol{m} \in \mathcal{E}(\mathcal{P}', \boldsymbol{W}) \text{ s.t. } m_p = i \overset{\text{Lemma 4.2.1}}{\iff} \mathcal{E}(\mathcal{P}' \setminus \{p\}, \boldsymbol{W} - i\boldsymbol{S}_p) \neq \emptyset \iff$$

$$\iff \exists \boldsymbol{x} \in \mathbb{N}^n \text{ s.t. } \boldsymbol{S}\boldsymbol{x} = \boldsymbol{W} - i\boldsymbol{S}_p \text{ and } \forall q \in \mathcal{P} \setminus (\mathcal{P}' \setminus \{p\}), x_q = 0$$

It is clear that the last statement is equivalent to the feasibility of ILP (4.25). □

Therefore the convolution algorithm can be compared to a recursive walk on an MDD that encodes the reachability set of the SPN where, instead of simply following arcs in the MDD, a feasibility of ILP of the form (4.25) is solved for each possible marking of the considered place (up to the bound of the marking of the place). Based on these considerations, for S-invariant reachable SPNs with known place marking bounds, it follows that the convolution algorithm can be used to generate an MDD encoding of its reachability set.

Checking S-invariant reachability condition As mentioned before, the S-invariant reachability condition on the SPNs is needed for the convolution algorithm to be able to decompose the reachability set by conditioning on the number of tokens in a place. In contrast, an MDD that encodes the reachability set of an SPN already defines such decomposition by its very structure; the S-invariant reachability condition is thus not needed for MDD-rec. Thus, for SPNs with finite reachability sets, MDD-rec is more general than the convolution algorithm because it can handle any product-form SPN with finite state space, in principle. In practice, the set of SPNs that MDD-rec can handle is the set of SPNs for which the MDDs encoding the reachability set can be generated in reasonable time. Furthermore, to the best of our knowledge there is no known algorithm that can decide S-invariant reachability for a general SPN without generating its reachability set, which further limits the practical scope of the convolution algorithm.

4.3 Efficient computation of performance measures

In this section we consider computation of performance measures. First, we note that it is very easy to modify MDD-rec to compute the probability $\Pr\{m_j = k\}$ that some place P_j contains $k \in \mathbb{N}$ tokens: in line 5 of Algorithm 6, if $j = l$ then perform the sum over the single index $s_l \in \{\phi_l(k)\}$; otherwise perform the sum as stated in the algorithm. The unnormalised probability $G\Pr\{m_j = k\}$ is obtained in this manner, from which the sought probability is obtained by multiplying with $1/G$. The following performance measures can now be easily computed:

- average number $n(P_j)$ of tokens in place P_j, $n(P_j) = \sum_{k=1}^{B_j} k\Pr\{m_j = k\}$ where B_j is the bound on the number of tokens in place P_j;
- utilization $u(P_j)$ of place P_j (probability that place P_j is nonempty), $u(P_j) = 1 - \Pr\{m_j = 0\}$.

Similarly, to compute the probability $\Pr\{e_j \geq k\}$ that transition T_j is enabled with enabling degree equal to or larger than some $k \in \mathbb{N}$, we modify line 5 of Algorithm 6 so that the sum is always taken over indices $s_l \in \{\phi_l(i) : i \geq kI_l(T_j)\}$; this is the set of renamed markings of place P_l for which the transition T_j can possibly have enabling degree at least k, depending on the markings of other places. We again obtain the unnormalised probability and multiply it with $1/G$ as before. From this it is easy to obtain the probability $\Pr\{e_j = k\}$ that transition T_j is enabled with enabling degree exactly k: $\Pr\{e_j = k\} = \Pr\{e_j \geq k\} - \Pr\{e_j \geq k + 1\}$. Now, assuming for example that all transitions have IS firing semantics, we easily compute the following performance measures:

- utilization $u(T_j)$ of transition T_j (probability $\Pr\{e_j > 0\}$ that transition T_j is enabled), $u(T_j) = \Pr\{e_j \geq 1\}$;
- throughput $x(T_j)$ of transition T_j (average number of firings of T_j in a unit of time), $x(T_j) = \sum_{k=1}^{\bar{E}_j} kW(T_j)\Pr\{e_j = k\}$, where $\bar{E}_j = \min_{1 \leq i \leq n}\{\lfloor B_i/I_i(T_j) \rfloor : I_i(T_j) > 0\}$ is an upper bound on the maximum enabling degree of transition T_j;

- throughput $x(P_j)$ of place P_j (average number of tokens that is removed from P_j in a unit of time), $x(P_j) = \sum_{T \in \mathcal{T}} I_j(T)x(T)$.

More complex performance measures that cannot be computed by a straightforward modification of MDD-rec could be computed by generating MDDs that encode subsets of the reachability set whose probability masses are needed in the computation of the sought performance measure. The unnormalised probability masses can then be computed by applying unmodified algorithm MDD-rec to the generated MDDs and then normalised using the normalising constant. MDDs as a data structure support many useful set and arithmetic operations that can be used in the generation of the needed MDDs.

MVA [23] is an algorithm for computation of performance measures of product-form SPNs. Compared with the methods proposed above, however, MVA has several disadvantages: 1) like convolution it assumes S-invariant reachability, in general requiring check of this condition by reachability set generation or manual proof, 2) it assumes SS firing semantics (i.e., it only allows marking-independent transition firing rates), and 3) it computes certain measures for all vectors $\boldsymbol{W}$ such that $\mathbf{0} \leq \boldsymbol{W} \leq \boldsymbol{V}$, where $\boldsymbol{V}$ is the load vector of a considered SPN, which makes its performance likely to be much worse than the algorithms proposed here which, due to equivalence of MDD-rec and convolution algorithm for S-invariant reachable SPNs, always perform a recursive walk over MDD nodes defined using only those vectors that are encountered during execution of the convolution algorithm.

4.4 Experiments

We have implemented the proposed algorithm MDD-rec for SPNs in C++, using library MEDDLY [41]. In our implementation, generation of the MDD encoding of the reachability set of the SPNs is handled by an efficient saturation algorithm [61] provided by MEDDLY. We have implemented labeling function L from the algorithm MDD-rec using the hashing-based map implementation `unordered_map` from the C++11 standard.

To compare performance with the convolution algorithm for SPNs, we have also implemented the convolution algorithm in C++. First the matrix $\boldsymbol{S}$ of minimal-support place invariants is computed using the standard algorithm [32]. Then the convolution algorithm, based on recurrence (4.11) (4.12) is performed. We use optimisation solver Gurobi [62] for the computation of marking sets by solving ILPs from the Proposition 4.2.2. We selected Gurobi based on its good performance on mixed integer linear programming benchmark problems [63]. In order to lower the number of ILPs that need to be constructed and solved during execution of the convolution algorithm, we implemented two optimisations based on place invariants: when computing a marking set $M_p(\mathcal{P}', \boldsymbol{W})$, we 1) search for a place invariant in matrix $\boldsymbol{S}$ of minimal-support place invariants that uniquely determines marking of place p (i.e., an invariant whose support contains p and all other places from the support have already been considered in ancestor recursive calls) and 2) if such invariant is not found, then we consider only possible markings ranging from 0 to a bound on the number of tokens in place p that is obtained from the invariants

in matrix $\boldsymbol{S}$ and the vector $\boldsymbol{W}$. These two optimisations result in a reduction on the scale of orders of magnitude in the number of considered ILPs and in correspondingly much shorter execution time of the convolution algorithm. Caching of values $G(\cdot, \cdot)$ computed during execution was also implemented using the C++11 `unordered_map`.

4.4.1 Performance comparison of MDD-rec and convolution

To test performance of the algorithms, we have adapted models from [64]. All the models that follow are S-invariant reachable SPNs with finite reachability sets, with exception of Example 3 model depicted in Fig. 4.3 which is only S-invariant reachable for parameter $k = 1$. For $k > 1$ this SPN is not S-invariant reachable because place P_3 can be marked only with numbers of tokens that are multiples of k—so that the reachability set is smaller than the one for $k = 1$—while the matrix of minimal-support S-invariants is the same as in case $k = 1$; thus in condition 4.5 only sufficiency (direction $\Rightarrow$) holds. In this case convolution algorithm cannot be used to compute the normalising constant, while MDD-rec can.

Fig. 4.1 shows an S-invariant reachable SPN parameterized by $n \in \mathbb{N}$ and composed of n identical subnets—first of which contains places P_1, P_2 and transitions T_1, T_2—connected in a series. Tokens in places $P_{2k}, k = 1, \ldots, n-1$ model resources that are reserved by the firing of transitions $T_{2k+1}, k = 1, \ldots, n-1$, respectively, and are released by the firing of transitions $T_{2k+2}, k = 1, \ldots, n-1$, respectively. The resulting net has $2n$ places and the same number of transitions. It was obtained by scaling a variant of the Example 1 net from [64].

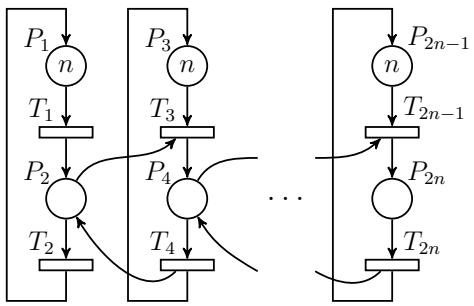


Figure 4.1: Example 1 SPN.

Likewise, Fig. 4.2 shows an S-invariant reachable SPN parameterized by n and consisting of the same n subnets which compete for central resources modeled by tokens in place P_0. This net has $2n + 1$ places and $2n$ transitions. This model was obtained by scaling the Example 2 net from [64].

In order to measure performance of the algorithms, we run them on the Example 1 and Example 2 SPNs for parameter n ranging from 5 to 100 with step 5. For both SPNs, this results in a range of models with reachability set cardinalities ranging from 252 for $n = 5$ to approximately 9.055×10^{58} for $n = 100$. In the MDDs and the convolution algorithm, places for Example 1 and Example 2 SPNs are taken in the orders in which they are numbered in the figures. For Example 1 and Example

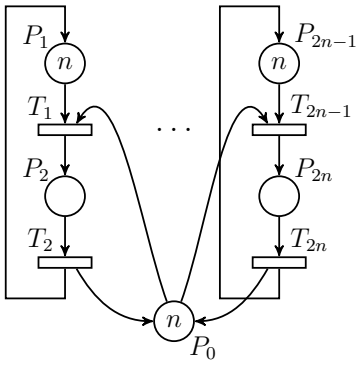


Figure 4.2: Example 2 SPN.

2 SPNs, Fig. 4.4 and 4.5, respectively, contain log-log plots of average times in seconds measured over 10 runs for the MDD generation, the proposed algorithm MDD-rec and the convolution algorithm, as functions of the reachability set size $|\mathcal{RS}|$. The error bars, too tight to see on the plots, represent 95% confidence intervals for the average times. In both cases, time taken for MDD-rec is orders of magnitude lower than the time taken for convolution algorithm; this is due to convolution algorithm needing to solve many ILPs during execution. Time taken to generate the MDDs is in both cases within an order of magnitude of the time taken for convolution. While both the convolution algorithm and the combined MDD generation + MDD-rec take a comparable amounts of time (note that the y-axis in the plots is logarithmic so that time needed for MDD generation can be taken as a good approximation of the total time needed for MDD generation and MDD-rec), MDD-rec is fully automatable and applicable to general SPNs while the convolution algorithm works only on S-invariant reachable SPNs and requires a separate proof of S-invariant reachability which to the best of our knowledge in the general case requires either generating the reachability set or performing a manual proof.

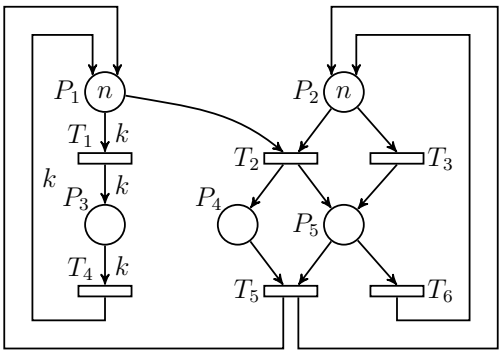


Figure 4.3: Example 3 SPN which is S-invariant reachable only for $k = 1$.

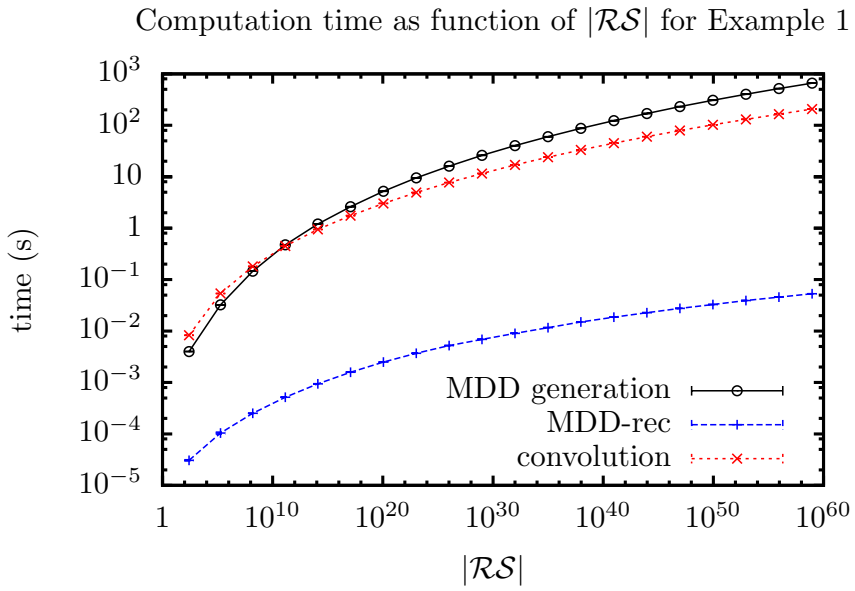


Figure 4.4: Measured times for Example 1 net.

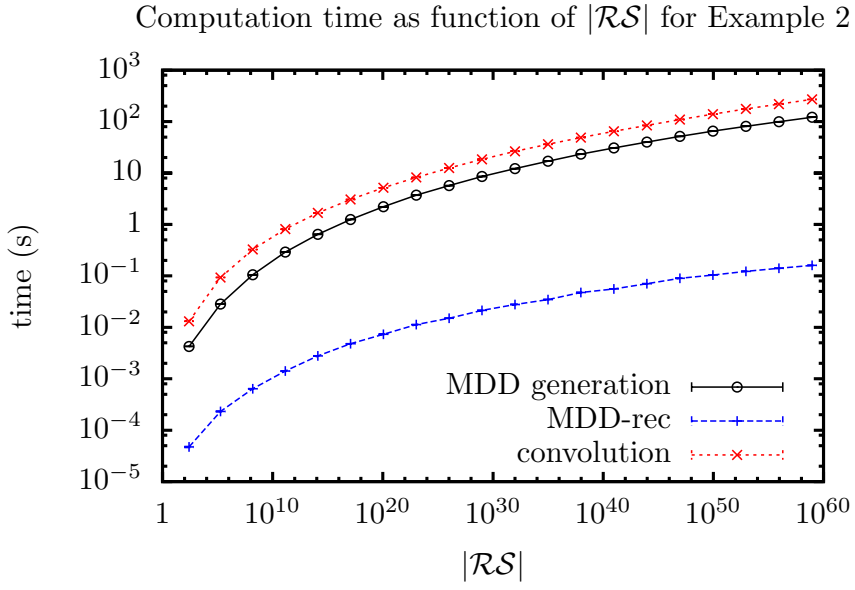


Figure 4.5: Measured times for Example 2 net.

4.4.2 Sensitivity to ordering of places

Performance of all three algorithms depends on the order in which SPN places are taken. We analyse this sensitivity to ordering of places on an SPN model adapted from Example 3 in [64] and reproduced here in Fig. 4.3. This SPN has only 5 places and it is thus possible to systematically test performance of the algorithms for all $5! = 120$ possible orderings of places. For $k = 1$ it is S-invariant reachable.

Fig. 4.6 shows a plot of average times in seconds over 10 runs for Example 3

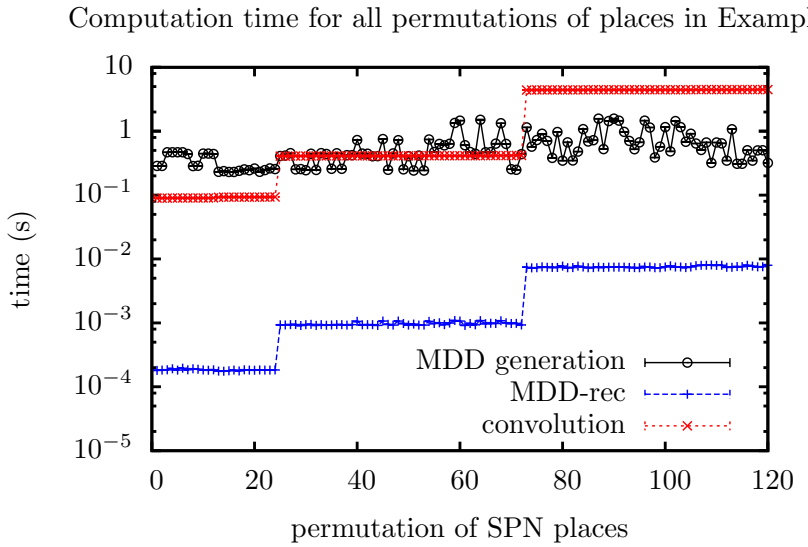


Figure 4.6: Measured times for Example 3 net ($n = 50$) and all place orderings.

class	**MDD nodes**	**ILPs solved**
$\{1, \ldots, 24\}$	157	1479
$\{25, \ldots, 72\}$	2757	6579
$\{73, \ldots, 120\}$	5307	72879

Table 4.1: Classes of place permutations for Example 3 SPN with $n = 50$.

SPN with $k = 1$ and $n = 50$ with error bars (again too tight to see on the plots) representing 95% confidence intervals for the average times. On y-axis are the times, and on the x-axis are all 120 possible permutations of places. For clarity, the permutations of places are sorted so that the time taken by convolution algorithm appears monotonically increasing on the plot.

We note that the set of all permutations of places can be partitioned into three equivalence classes with regard to the execution time of the convolution algorithm. These are depicted in Table 4.1 where **class** is the set of indices of permutations as depicted on the x-axis of Fig. 4.6, **MDD nodes** is the number of nodes in the MDD that encodes the reachability set of the SPN, and **ILPs solved** is the number of ILPs solved during execution of the convolution algorithm. The table shows that the number of nodes in the MDD over which MDD-rec walks and, equivalently, the same number of values $G(\cdot, \cdot)$ which the convolution algorithm computes are highly sensitive to the ordering of places, as well as the number of solved ILPs in convolution.

As can be expected, convolution algorithm and MDD-rec are fastest on the first class which has the lowest number of MDD nodes and are slowest on the third class with the highest number of MDD nodes. The times obtained for MDD generation are more variable due to the saturation algorithm that is used in reachability set generation which generates many intermediate MDDs. MDD generation is in some

cases faster and in some cases slower than the convolution algorithm. Additional measurements show that for $n = 10$ MDD generation is faster than convolution for all permutations of places, and for $n = 100$ it is slower than convolution for all permutations. MDD-rec is faster than both by orders of magnitude in all cases.

From the above considerations and other more limited experiments, we expect to find a similar situation for other models as well; performance of all algorithms will likely vary over orders of magnitude with the permutations of places, but the MDD-rec will always be much faster than the other two algorithms.

4.5 Summary

In this chapter we have proposed an algorithm MDD-rec for computation of normalising constant for product-form models with finite state spaces, and have presented related analysis methods for the computation of stationary performance indices.

In contrast to previously developed convolution algorithm for the computation of the normalising constant and MVA algorithm for computation of the stationary performance measures, the algorithm MDD-rec and the related analysis methods do not require the analysed product-form SPNs to be in the special class of S-invariant reachable SPNs, membership of which, at the state of the art, can in general only be tested computationally by the generation of the reachability set of the SPN.

We have tested the performance of the algorithm MDD-rec, showing that its performance is comparable to the performance of the convolution algorithm even when the generation of the reachability set is taken into account.

5

Conclusion

In this thesis we have presented two algorithms for stationary analysis of stochastic Petri nets: an algorithm for perfect sampling in SPNs and an algorithm for computation of the normalising constant for product-form SPNs. The algorithms use efficient data structures and methods—multi-valued decision diagrams and the saturation algorithm—in order to optimise the key step of generation of reachability sets of SPN models, allowing analysis of models with very large reachability sets.

5.1 Algorithm for perfect sampling in SPNs

First we have presented an algorithm for sampling from the stationary probability distribution of stochastic Petri nets, based on coupling from the past and efficient encoding of reachability sets and transition functions of the SPNs using MDDs. In contrast to previous approaches which require some constraints on the structure of the Petri net, the proposed algorithm can be applied to general stochastic Petri nets, requiring only the finiteness of the reachability set and some assumptions on the type of marking-dependence of firing rates of transitions. We have then presented spnps, a tool that implements the previously presented algorithm. By testing the tool on several SPNs we have shown that the algorithm generally performs well, in some cases allowing sampling from reachability sets with more than 10^{200} states in reasonable time. We have also evaluated the performance of the tool on SPN models of fork-join queueing networks with some resource competition. The results of our tests suggest that the algorithm as implemented is likely to be more efficient for sampling in models that are composed of many loosely coupled components.

Future work on this algorithm can be taken in several directions. First, it might be worthwhile to explore different MDD encodings of the SPN reachability set, as in [40], in order to improve running time of the algorithm. Second, the algorithm should be easy to adapt to other structured stochastic formalisms used in performance evaluation. Third, the notion of multimodal behaviour could be formalised and the spnps tool extended so that it computes some measures of multimodality. Fourth, the spnps tool could be integrated with one of the software tools that support SPN simulation.

5.2 Algorithm for computation of normalising constant for product-form models

We have also proposed algorithm MDD-rec for computation of normalising constant for product-form models with finite state spaces, based on efficient generation and encoding of the models' state spaces using MDDs. We have shown that MDD-rec is equivalent to convolution on S-invariant reachable product-form SPNs in terms of data flow but, unlike convolution, it does not require S-invariant reachability and is therefore more general. We have also proposed related methods for computation of performance measures for product-form SPNs. These methods are more general than the MVA algorithm which shares with convolution the requirement of S-invariant reachability and in addition requires marking-independence of transition firing rates. Finally, we have implemented the proposed algorithm MDD-rec for SPNs and compared its performance to the convolution algorithm on two example models, showing that the performance of the algorithms is comparable.

Similarly to the perfect sampling algorithm, future work could be done on different MDD encodings of the reachability set with the aim of improving the run time of the algorithm. Also, the algorithm could be generalised to models with product-forms that include factors which depend on states of multiple submodels. Finally, for product-form models with infinite state spaces, normalising constant could be approximated by applying MDD-rec to a finite subset of the state space obtained by bounding the model, where this subset is selected so that its probability mass is close to 1.

Bibliography

[1] S. Balsamo, A. Marin, and I. Stojic, "Perfect sampling in stochastic Petri nets using decision diagrams," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pp. 126–135, Oct 2015.

[2] S. Balsamo, A. Marin, and I. Stojic, "Spnps: A tool for perfect sampling in stochastic Petri nets," in *Quantitative Evaluation of Systems (QEST), 2016 13th International Conference on*, 2016.

[3] S. Balsamo, A. Marin, and I. Stojic, "Testing spnps perfect sampling tool on fork-join queueing networks," in *Performance Evaluation Methodologies and Tools (VALUETOOLS) 2016, 10th International Conference on*, 2016.

[4] S. Balsamo, A. Marin, and I. Stojic, "Computation of normalising constant for product-form models using decision diagrams," *Performance Evaluation*, submitted.

[5] S. Balsamo, A. Marin, and I. Stojic, "Optimisation of servers with different quality of services," in *2014 IEEE 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems*, pp. 142–151, Sept 2014.

[6] S. Balsamo, A. Marin, and I. Stojic, "Deriving the performance indices in product-form stochastic Petri nets: open problems and simulation," in *EUROSIS European Simulation Multiconference*, 2015.

[7] M. K. Molloy, "Performance analysis using stochastic Petri nets," *IEEE Trans. on Comput.*, vol. 31, no. 9, pp. 913–917, 1982.

[8] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic Petri nets*. Wiley, 1995.

[9] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[10] W. Henderson, D. Lucic, and P. G. Taylor, "A net level performance analysis of Stochastic Petri Nets," *J. Austral. Math. Soc. Ser. B*, vol. 31, pp. 176–187, 1989.

[11] A. A. Lazar and T. G. Robertazzi, "Markovian Petri Net Protocols with Product Form Solution.," *Perf. Eval.*, vol. 12, no. 1, pp. 67–77, 1991.

[12] S. Haddad, J. Mairesse, and H. Nguyen, "Synthesis and analysis of product-form Petri nets," *Fundam. Inform.*, vol. 122, no. 1-2, pp. 147–172, 2013.

[13] G. Balbo, S. C. Bruell, and M. Sereno, "Product form solution for Generalized Stochastic Petri Nets," *IEEE Trans. on Software Eng.*, vol. 28, no. 10, pp. 915–932, 2002.

[14] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: theory and applications," *Multiple-Valued Logic*, vol. 4, no. 1, pp. 9–62, 1998.

[15] A. S. Miner and G. Ciardo, "Efficient reachability set generation and storage using decision diagrams," in *Application and Theory of Petri Nets 1999* (S. Donatelli and J. Kleijn, eds.), vol. 1639 of *Lecture Notes in Computer Science*, pp. 6–25, Springer Berlin Heidelberg, 1999.

[16] G. Ciardo, G. Lüttgen, and R. Siminiceanu, "Efficient symbolic state-space construction for asynchronous systems," in *Application and Theory of Petri Nets 2000* (M. Nielsen and D. Simpson, eds.), vol. 1825 of *Lecture Notes in Computer Science*, pp. 103–122, Springer Berlin Heidelberg, 2000.

[17] J. G. Propp and D. B. Wilson, "Exact sampling with coupled Markov chains and applications to statistical mechanics," *Random Struct. Algorithms*, vol. 9, no. 1-2, pp. 223–252, 1996.

[18] A. Bušić, B. Gaujal, and J.-M. Vincent, "Perfect simulation and non-monotone Markovian systems," in *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '08, (ICST, Brussels, Belgium), pp. 27:1–27:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[19] A. Bušić, B. Gaujal, and F. Perronnin, "Perfect sampling of networks with finite and infinite capacity queues," in *Analytical and Stochastic Modeling Techniques and Applications* (K. Al-Begain, D. Fiems, and J.-M. Vincent, eds.), vol. 7314 of *Lecture Notes in Computer Science*, pp. 136–149, Springer Berlin Heidelberg, 2012.

[20] A. Bouillard, A. Bušić, and C. Rovetta, "Perfect sampling for closed queueing networks," *Performance Evaluation*, vol. 79, pp. 146 – 159, 2014. Special Issue: Performance 2014.

[21] A. Bouillard and B. Gaujal, "Backward coupling in Petri nets," in *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, (New York, NY, USA), ACM, 2006.

[22] J. Coleman, W. Henderson, and P. Taylor, "Product form equilibrium distributions and a convolution algorithm for stochastic Petri nets," *Performance Evaluation*, vol. 26, no. 3, pp. 159 – 180, 1996.

[23] M. Sereno and G. Balbo, "Mean value analysis of stochastic Petri nets," *Perform. Eval.*, vol. 29, pp. 35–62, Feb. 1997.

[24] C. A. Petri, "Communication with automata," *Technical Report RADC-TR-65-377*, 1966.

[25] T. Agerwala, "Special feature: Putting Petri nets to work," *IEEE Computer*, vol. 12, pp. 85–94, Dec 1979.

[26] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[27] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic well-formed colored nets and symmetric modeling applications," *IEEE Transactions on Computers*, vol. 42, pp. 1343–1360, Nov 1993.

[28] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," in *International Workshop on Timed Petri Nets*, (Washington, DC, USA), pp. 106–115, IEEE Computer Society, 1985.

[29] M. Ajmone Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Trans. Comput. Syst.*, vol. 2, pp. 93–122, May 1984.

[30] G. Balbo, "Introduction to Generalized Stochastic Petri Nets," in *Proceedings of the 7th International Conference on Formal Methods for Performance Evaluation*, SFM'07, (Berlin, Heidelberg), pp. 83–131, Springer-Verlag, 2007.

[31] G. Ciardo, "Toward a definition of modeling power for stochastic Petri net models," tech. rep., Durham, NC, USA, 1987.

[32] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*. Springer Publishing Company, Incorporated, 2nd ed., 2010.

[33] G. Ciardo, J. Muppala, and K. S. Trivedi, "On the solution of GSPN reward models," *Perform. Eval.*, vol. 12, pp. 237–253, Aug. 1991.

[34] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, Aug 1986.

[35] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on*, pp. 428–439, Jun 1990.

[36] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle, "On the use of MTBDDs for performability analysis and verification of stochastic systems," *The Journal of Logic and Algebraic Programming*, vol. 56, no. 1, pp. 23 – 67, 2003.

[37] A. S. Miner, "Efficient solution of GSPNs using canonical matrix diagrams," in *Petri Nets and Performance Models, 2001. Proceedings. 9th International Workshop on*, pp. 101–110, 2001.

[38] G. Ciardo, G. Lüttgen, and R. Siminiceanu, "Saturation: An efficient iteration strategy for symbolic state-space generation," in *Tools and Algorithms for the Construction and Analysis of Systems* (T. Margaria and W. Yi, eds.), vol. 2031 of *Lecture Notes in Computer Science*, pp. 328–342, Springer Berlin Heidelberg, 2001.

[39] G. Ciardo, R. Marmorstein, and R. Siminiceanu, "Saturation unbound," in *Tools and Algorithms for the Construction and Analysis of Systems* (H. Garavel and J. Hatcliff, eds.), vol. 2619 of *Lecture Notes in Computer Science*, pp. 379–393, Springer Berlin Heidelberg, 2003.

[40] G. Ciardo, "Reachability set generation for Petri nets: Can brute force be smart?," in *Applications and Theory of Petri Nets 2004* (J. Cortadella and W. Reisig, eds.), vol. 3099 of *Lecture Notes in Computer Science*, pp. 17–34, Springer Berlin Heidelberg, 2004.

[41] J. Babar and A. S. Miner, "Meddly: Multi-terminal and edge-valued decision diagram library," in *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010*, pp. 195–196, 2010.

[42] A. Miner, "Saturation for a general class of models," in *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pp. 282–291, Sept 2004.

[43] P. Haas, *Stochastic Petri Nets: Modelling, Stability, Simulation*. Springer Series in Operations Research, Springer, 2002.

[44] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd ed., 1999.

[45] S. Tani, K. Hamaguchi, and S. Yajima, "The complexity of the optimal variable ordering problems of shared binary decision diagrams," in *Algorithms and Computation* (K. Ng, P. Raghavan, N. Balasubramanian, and F. Chin, eds.), vol. 762 of *Lecture Notes in Computer Science*, pp. 389–398, Springer Berlin Heidelberg, 1993.

[46] K. B. Athreya and S. N. Lahiri, *Measure Theory and Probability Theory*. Springer-Verlag New York, 2006.

[47] F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," *ACM Trans. Math. Softw.*, vol. 32, pp. 1–16, Mar. 2006.

[48] M. Weber and E. Kindler, "The Petri net markup language," in *Petri Net Technology for Communication-Based Systems* (H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, eds.), vol. 2472 of *Lecture Notes in Computer Science*, pp. 124–144, Springer Berlin Heidelberg, 2003.

[49] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia, "Petri net analysis using boolean manipulation," in *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, (London, UK, UK), pp. 416–435, Springer-Verlag, 1994.

[50] P. Heidelberger and K. Trivedi, "Analytic queueing models for programs with internal concurrency," *IEEE Trans. Comput.*, vol. C-32, pp. 73–82, 1983.

[51] R. Chen, "An upper bound solution for homogeneous fork/join queueing systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 874–878, 2011.

[52] S. Balsamo, P. G. Harrison, and A. Marin, "Methodological Construction of Product-form Stochastic Petri-Nets for Performance Evaluation," *J. of System and Software*, vol. 85, no. 7, pp. 1520–1539, 2012.

[53] H. Dai, "Exact simulation for fork-join networks with heterogeneous service," *Int. J. of Statistics and Probability*, vol. 4, no. 1, 2015.

[54] A. Bouillard and B. Gaujal, "Backward coupling in bounded free-choice nets under Markovian and non-Markovian assumptions," *Discrete Event Dynamic Systems*, vol. 18, no. 4, pp. 473–498, 2008.

[55] J. Desel and J. Esparza, *Free choice Petri nets*. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press, 1995.

[56] J. R. Jackson, "Jobshop-like queueing systems," *Management Science*, vol. 10, pp. 131–142, 1963.

[57] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *J. ACM*, vol. 22, no. 2, pp. 248–260, 1975.

[58] J. P. Buzen, "Computational algorithms for closed queueing networks with exponential servers," *Commun. ACM*, vol. 16, pp. 527–531, Sept. 1973.

[59] M. Sereno and G. Balbo, "Computational algorithms for product form solution stochastic Petri nets," in *Petri Nets and Performance Models, 1993. Proceedings., 5th International Workshop on*, pp. 98–107, Oct 1993.

[60] A. Schrijver, *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[61] A. S. Miner, "Saturation for a general class of models," *IEEE Transactions on Software Engineering*, vol. 32, pp. 559–570, Aug 2006.

[62] Gurobi Optimization, Inc., "Gurobi optimizer reference manual." `http://www.gurobi.com`, 2016. [online; accessed 15/8/2016].

[63] H. Mittelmann, "Decision tree for optimization software." `http://plato.asu.edu/bench.html`, 2016. [online; accessed 15/8/2016].

[64] J. L. Coleman, "Algorithms for product-form stochastic Petri nets-a new approach," in *Petri Nets and Performance Models, 1993. Proceedings., 5th International Workshop on*, pp. 108–116, Oct 1993.